1. Introduction
    1. 90 = 100: A Proof
    2. Introduction: logic motivation
    3. Introduction: logic definition
2. Propositional Logic
    1. A formal vocabulary
        1. propositions
        2. formulas
    2. Reasoning with truth tables
        1. Propositional Logic: truth tables
        2. Propositional Logic: limitations of truth tables
    3. Reasoning with equivalences
        1. Propositional Logic: equivalences
        2. Propositional Logic: normal forms
        3. soundness and completeness
    4. Reasoning with inference rules
        1. Propositional Logic: inference rules
        2. using subproofs
        3. Propositional Logic: soundness and completeness revisited
        4. Propositional Logic: type checking
        5. Propositional Logic: conclusions
    5. Exercises for Propositional Logic I
    6. Exercises for Propositional Logic II
3. Relations and Models
    1. Relations and Logic: using relations
    2. properties of relations
    3. interpretations
    4. Relations and Logic: Non-standard Interpretations
    5. modeling with relations
4. First-Order Logic

90 = 100: A Proof

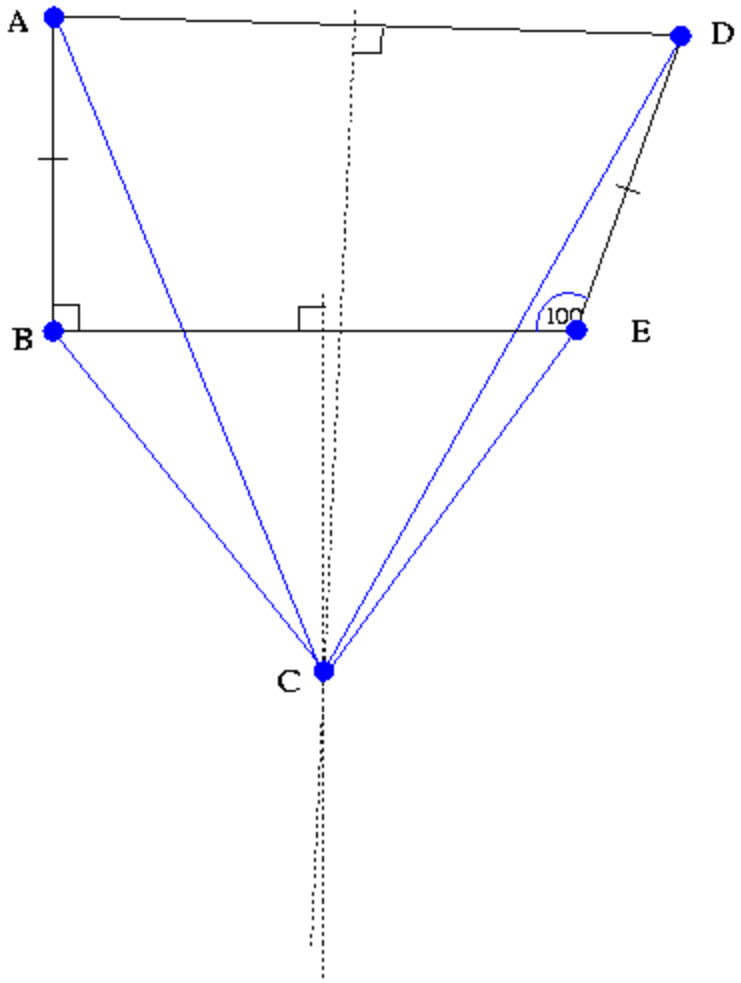
Construct a four-sided figure ABED as follows:

- $|\angle ABE| = 90°$
- $|\angle DEB| = 100°$
- $|AB| = |ED|$

Using that as a starting point, we now tinker a bit to show that 90=100:

- Draw the perpendicular bisectors to BE and AD; call the point where they meet “C”.

Note: Actually, we must prove that those two perpendicular bisectors really do meet at all (i.e., that the point C even exists). In this case, it turns out to be pretty clear — it's not hard to argue that lines AD and BE aren't parallel, and therefore their perpendicular bisectors aren't parallel, and so they must intersect (in Euclidean geometry). Still, be alert for people making glib assertions in proofs.



A construction to help prove that 90=100

Looking at this figure, some warning flags should be going up: How do we know C lies **below** BD? Might it lie above BD? Or exactly on BD? It turns out that the argument below is the same in all of these cases, though you'll certainly want to verify this to yourself later.

1	$\lvert AB\rvert = \lvert ED\rvert$	By construction.
2	$\lvert BC\rvert = \lvert EC\rvert$	C is on the perpendicular bisector of BE (thus $\triangle BEC$ is isosceles).
3	$\angle CBE \cong \angle BEC$	Base angles of isosceles triangle BEC are congruent.
4	$\lvert \angle CBE\rvert = \lvert \angle BEC\rvert$	Congruent angles have equal measures; line 3.
5	$\lvert AC\rvert = \lvert DC\rvert$	C is on the perpendicular bisector of AD (thus $\triangle ADC$ is isosceles).
6	$\triangle ABC \cong \triangle DEC$ (!!)	Triangles with three congruent sides are congruent (Euclid's Side-Side-Side congruence theorem); lines 1,2,5.
7	(From here, it's just routine steps to conclude 90=100:)	
8	$\angle ABC \cong \angle DEC$	Corresponding parts of congruent triangles are congruent; line 6.
9	$\lvert \angle ABC\rvert = \lvert \angle DEC\rvert$	Congruent angles have equal measures; line 8.
10	$\lvert \angle ABC\rvert = \lvert \angle ABE\rvert + \lvert \angle CBE\rvert$	By construction.
11	$\lvert \angle DEC\rvert = \lvert \angle DEB\rvert + \lvert \angle BEC\rvert$	By construction.

12	|∠DEC| = |∠DEB| + |∠CBE|	Substituting equals with equals; lines 11 and 4.
13	|∠ABC| = |∠DEB| + |∠CBE|	Substituting equals with equals; lines 12 and 9.
14	|∠ABE| + |∠CBE| = |∠DEB| + |∠CBE|	Substituting equals with equals; lines 13 and 10.
15	|∠ABE| = |∠DEB|	Subtracting equals from equals remains equal.
16	90 = |∠DEB|	By construction, and substituting equals with equals; line 15.
17	90 = 100	By construction, and substituting equals with equals; line 16.

A useful corollary: 0=1.

1	90 = 100	Previous theorem.
2	0 = 10	Subtracting equals (90) from equals remains equal.
3	0 = 1	Dividing equals by non-zero equals (10) remains

		equal.

Exercise:

Problem:

If you feel this result is incorrect, then the challenge for you is to find the first line which is false.

Solution:

The flaw is extremely hard to find. We won't actually give the solution, but here's a hint on how to go about attacking the puzzle:

Note that finding the bug in the proof is the same skill as debugging a program. A good approach is to try various degenerate inputs. In this case, there are a couple of “inputs” to the construction—the length of CD is arbitrary; no matter how long or short the proof should apply equally well. Similarly, the angle 100° seems arbitrary; fiddling with inputs like these (making them very small or very large) might give you some clues as to where the bug is. A **very** careful drawing will clear things up.

You may have noticed that the proof given here has some very minuscule steps—e.g. “Congruent angles have equal measure.” Usually such simple steps can be omitted, since they are obvious to any reader. We include them for a few reasons:

- As a careful thinker, you should recognize that such small steps really are part of the complete reasoning, even if they're not worth mentioning continually.
- If a computer is checking a proof, it needs to actually include those steps.
- Programmers do need to be concerned with distinctions about (abstract) types—the difference between angles and their measures, in this case.

- Sometimes a line's justification is glibly given as “by construction”, when that may not even be correct `!-)`.

In this course, we'll spend a few weeks working with proofs which **do** include all the small, pedantic steps, to instill a mental framework for what a rigorous proof is. But after that, you can relax your proofs to leave out such low-level steps, once you appreciate that they are being omitted.

Introduction: logic motivation

The ancient Greeks loved to hang around on the stoa, sip some wine, and debate. But at the end of the day, they wanted to sit back and decide who had won the argument. When Socrates **claims** that one statement follows from another, is it actually so? Shouldn't there be some set of rules to officially determine when an argument is correct? Thus began the formal study of logic.

> **Note:** The three fundamental studies were the Trivium — grammar (words), logic (reasoning), and rhetoric (effective communication). These allowed study of the Quadrivium — arithmetic (patterns in number), geometry (patterns in space), music (patterns in tone), and astronomy (patterns in time). All together, these subjects comprise the seven liberal arts.

These issues are of course still with us today. And while it might be difficult to codify real-world arguments about (say) gun-control laws, programs **can** be fully formalized, and correctness can be specified. We'll look at three examples where formal proofs are applicable:

- playing a simple game, WaterWorld;
- checking a program for type errors;
- circuit verification.

Many other areas of computer science routinely involve proofs, although we won't explore them here. Manufacturing robots first prove that they can twist and move to where they need to go before doing so, in order to avoid crashing into what they're building. When programming a collection of client and server computers, we usually want to prove that the manner in which they communicate guarantees that no clients are always ignored. Optimizing compilers prove that, within your program, some faster piece of code behaves the same as and can replace what you wrote. With software

systems controlling more and more life-critical applications, it's important to be able to **prove** that a program always does what it claims.

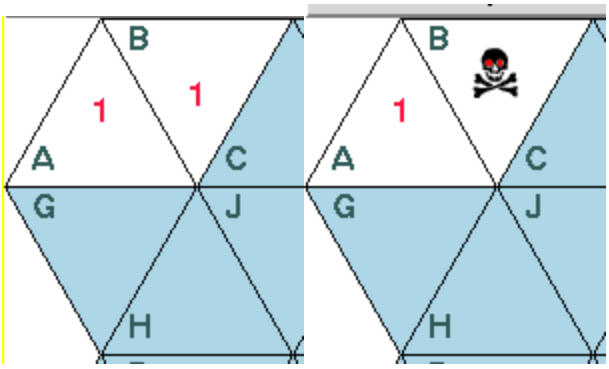
WaterWorld

Consider a game called WaterWorld, where each location is either empty sea or contains a pirate. When you enter a location, you must **correctly** anticipate whether or not it contains pirates.

- If you correctly anticipate open sea, you are able to enter and determine how many of the (up to 3) adjacent locations contain a pirate.
- If you correctly anticipate a pirate, the location is tagged as dangerous, and you gather no further information.

Furthermore, there are really two types of moves: guesses, and assertions. If you make an assertion, then even if you happen to be correct but it is possible you could have been wrong, then it is an error. Also, it is an error if you make a guess about a location if it is actually possible to assert a location's contents. The interesting fact about these types of games is that while sometimes guesses are necessary (when?), surprisingly often an assertion can be made.

(You can freely download WaterWorld.)



Glimpses of two different
WaterWorld boards

For instance, in the first board, what assertions can we be sure of? What, exactly, is your reasoning? How about in the second board? You can certainly envision wanting a computer player that can deduce certain moves, an make those for you automatically.

Type Checking

When writing a program, we'd like to simply look at the program and determine whether it has any bugs, without having to run it. We'll see in the future, however, that such a general problem cannot be solved. Instead, we focus on finding more limited kinds of errors. **Type checking** determines whether all functions are called with the correct **type** of inputs. E.g., the function `+` should be called with numbers, not Booleans, and a function which a programmer has declared to return an integer really should always return an integer. Consider the following program:

```
// average:
// Simply divide sum by N, but guard against
dividing by 0.
//
real-or-false average( real sum, natNum N ) {
  if (N != 0)
    return sum / N;
  else
    return false;
}
```

One reason programmers are required to declare the intended type of each variable is so that the computer (the compiler) can **prove** that certain errors won't occur. How can you or the compiler prove, in the above, that

`average` returns a real number or false, but never returns (say) a string, and doesn't raise an exception? Deductions are made based on premises about the types that are passed in, along with axioms about the input and return types of the built-in functions `if`, `!=`, and `/`, as well as which exceptions those built-ins might raise.

Consider this variant:

```
// augment-average:
// Given an old sum and N, compute the average if
one more
// datum were included.
//
real augment_average( real old_sum, natNum old_N,
real new_datum ) {
  return average( old_sum + new_datum, old_N + 1
);
}
```

Most compilers will reject `augment-average`, claiming that it may actually return `false`. However, we're able prove that it really will only return a `real`, by using some knowledge about natural numbers and adding 1, plus some knowledge of what `average` returns. (Note that our reasoning uses aspects of `average`'s interface which aren't explicitly stated; most type systems aren't expressive enough to allow more detailed type contracts, for reasons we'll allude to later.) So we see that many compilers have overly conservative type-checkers, rejecting code which is perfectly safe, because they are reasoning with only a narrow set of type-rules.

This example alludes to another use of logic: Not only is it the foundation of writing proofs (ones that can be created or checked by computers), but logic can also be used as an **unambiguous specification language**. Observe that while a function's implementation is always specified formally and unambiguously — in a programming language — the interface is specified entirely English, aside from a few type declarations. Many bugs

stem from ambiguities in the English, that different humans interpret differently (or, don't think about). Being able to use logic to specify an interface (and cannot be modified even if the somebody later tunes the implementation) is an important skill for programmers, even when those logic formulas aren't used in proofs.

Circuit Verification

Given a circuit's blueprints, will it work as advertised? In 1994, Intel had to recall five million of its Pentium processors, due to a bug in the arithmetic circuitry: This cost Intel nearly **half a billion** dollars, lots of bad publicity, and it happened **after** intensive testing. Might it have been possible to have a **program** try to prove the chip's correctness or uncover an error, before casting it in silicon?

Software and hardware companies are increasingly turning to the use of automated proofs, rather than semi-haphazard testing, to verify (parts of) large products correct. However, it is a formidable task, and how to do this is also an active area of research.

There are of course many more examples; one topical popular concern is verifying certain security properties of electronic voting machines (often provided by vendors who keep their source software a proprietary secret).

Having proofs of correctness is not just comforting; it allows us to save effort (less time testing, and also able to make better optimizations), and prevent recall of faulty products. But: who decides a proof is correct — the employee with best SAT scores?!? Is there some trusted way to verify proofs, besides careful inspection by a skilled, yet still error-prone, professional?

"Many highly intelligent people are poor thinkers. Many people of average intelligence are skilled thinkers. The power of the car is separate from the way the car is driven." *Edward De Bono, consultant, writer, and speaker (1933-)*

Introduction: logic definition

What are proofs? (informal)

Example:
The following submission from an anonymous engineer to the January, 1902 edition of Popular Mechanics caught my eye. Seems like something every Boy/Girl Scout and Architect should know.
“HOW TO USE THE WATCH AS A COMPASS: Very few people are aware of the fact that in a watch they are always provided with a compass, with which, when the sun is shining, the cardinal points can be determined. All one has to do is to point the hour hand to the sun and south is exactly half way between the hour and the figure 12 on the watch. This may seem strange to the average reader, but it is easily explained. While the sun is passing over 180 degrees (east to west) the hour hand of the watch passes over 360 degrees (from 6 o'clock to 6 o'clock). Therefore the angular movement of the sun in one hour corresponds to the angular movement of the hour hand in half an hour; hence, if we point the hour hand toward the sun the line from the point midway between the hour hand and 12 o'clock to the pivot of the hands will point to the south. — Engineer.”

They give an argument of correctness; is that really a proof? Well, there are some ambiguities: Do I hold the watch vertically, or, in the plane of the sun's arc? Certainly I can't hold it up-side down, even though this isn't explicitly stated. Furthermore, the correctness of the reasoning relies on some unstated assumptions. E.g., the sun is at its highest (northernmost) point of its transit at noon. Is this actually true? Does it depend on the time of year? I'm not exactly sure (and will have to sit down and scratch my head and draw pictures of orbits, to convince myself). Certainly there are at least a couple of caveats: even beyond account for Daylight Savings Time, the solar-time and clock-time only align at time-zone boundaries, and they drift up to an hour apart, before the next boundary rectifies the difference. Is this presuming I'm in the northern hemisphere? What if I'm **on** the equator?

To be fair, the intent of this anecdote was to give enough evidence to convince you, not necessarily to be a complete, stand-alone self-contained proof. But in writing out a careful proof, one is forced to consider all the points just made; being forced to understand these can lead you to better understand the procedure yourself. But be careful to distinguish between something which sounds reasonable, and something that you're certain of.

An argument by form

How can we tell true proofs from false ones? What, exactly, are the rules of a proof? These are the questions which will occupy us.

Proofs are argument by form. We'll illustrate this with three parallel examples of a particular proof form called **syllogism**.

Example:

1	All people are mortal.	Premise
2	Socrates is a person.	Premise
3	Therefore, Socrates is mortal.	Syllogism, lines 1,2

Example:

1	All [substitution ciphers] are [vulnerable to brute-force attacks].	Premise
2	The [Julius Caesar cipher] is a [substitution cipher].	Premise
3	Therefore, the [Julius Caesar cipher] is [vulnerable to brute-force attacks].	Syllogism, lines 1,2

Note that you don't need to know anything about cryptography to know that the conclusion follows from the two premises. (Are the premises indeed true? That's a different question.)

Example:

1	All griznoxes chorble happily.	Premise
2	A floober is a type of griznox.	Premise
3	Therefore, floobers chorble happily.	Syllogism, lines 1,2

You don't need to be a world-class floober expert to evaluate this argument, either.

Note: Lewis Carroll, a logician, has developed many whimsical examples of syllogisms and simple reasoning. (Relatedly, note how the social context of Carroll's examples demonstrates some feminist issues in teaching logic.)

As you've noticed, the **form** of the argument is the same in all these. If you are assured that the first two premises are true, then, without any true understanding, you (or a computer) can automatically come up with the conclusion. A syllogism is one example of a **inference rule** — that is, a rule form that a computer can use to deduce new facts from known ones.

Some non-proofs

Of course, not all arguments are valid proofs. Identifying invalid proofs is just as interesting as identifying valid ones.

Note: `Homer: Ah, not a bear in sight. The Bear Patrol must be working. Lisa: That's specious reasoning, Dad. Homer: Thank you, honey. Lisa: By your logic, this rock keeps tigers away. Homer: Oh? How does it work? Lisa: It doesn't work. Homer: Uh-huh. Lisa: It's just a stupid rock. Homer: Uh-huh. Lisa: But I don't see any tigers around here, do you? [pause] Homer: Lisa, I want to buy your rock! [A moment's hesitation … and money changes hands.]` (From The Simpsons *Much Apu About Nothing*.)

If Lisa isn't around, who will identify specious reasoning for us? We can certainly use her approach of finding other particular examples that follow the same argument, yet lead to a clearly erroneous conclusion.

Example:
Suppose that my friend makes the following argument:

1	Warm cola tastes bad.	Premise
2	Warm salt-water tastes bad.	Premise
3	Therefore, mixing them together tastes bad.	“Common-sense conclusion”, lines 1,2

I'm skeptical, so I have a sip; sure enough, the conclusion is indeed true. But is the **proof** correct — does the “common-sense conclusion” rule actually hold? In order to refute the form of the argument, we can try similar arguments which have the same form but a false conclusion (as Lisa did).

1	Ice-cold coke tastes good.	Premise
2	Ice coffee tastes good.	Premise
3	Therefore, mixing them together tastes good.	“Common-sense conclusion”, lines 1 and 2.

After another unfortunate sip, I verify that this conclusion is not true, and therefore my friend's reasoning is at fault.
My friend responds by claiming that the “common-sense conclusion” is too valid; the rule is that bad-taste is preserved upon mixing, not that any taste is preserved. While I'm inclined to believe that, we realize we can still test this more refined rule: can you come up with an instance of mixing together bad-tasting things and ever getting a yummy result? (Say, salt and flour, which can be mixed and baked to get delicious saltines! The argument continues, about whether the form of the argument precludes baking, and so on.)

> The end result (after I take some antacid) is that we have a clearer understanding of the initially vague “common-sense conclusion”, and stricter rules about when it applies. Thus, refining the argument has led us to a greater understanding.

The above examples are a bit frivolous, but the procedure of looking for counterexamples applies to many real-world dilemmas. It also highlights the difference between a correct proof, and a faulty proof that might still happen to lead to a true result. (By the way, this is the exact same skill used when trying to come up with an algorithm for a problem: “well, the algorithm works for this input, but can I find a something that makes one of the steps fail?” If so, you then try refining your algorithm “well, I can add a test to take care of that problem; is that enough so that it always works?”)

Exercise:

> **Problem:**
>
> Solve this statement for [X]: "It is wrong to ban [X]. Such a ban would punish those reasonable citizens who would use [X] responsibly, while those who really want to abuse [X] will be able to get it anyway, through a black market which will only subsidize other criminal activities."

> **Solution:**
>
> This argument is or has been commonly used for varying topics —
>
> - marijuana,
> - alcohol,
> - all drugs,
> - handguns,
> - birth control,
> - prostitution,
> - encryption technology.

> The interesting part, is that the traditional Left and Right political positions **each** use this argument for some of these items, while rejecting the argument when used for other items.
>
> A more rational response is to either accept all the above, or none of the above, **or** to realize that the stated argument wasn't everything — that there might be implicit assumptions or arguments which actually do distinguish between these cases (the different interpretations of "[X]"). Being able to articulate the differences is essential. The more refined arguments may be more nuanced, and less able to fit into a sound-bite, but lead to a better understanding of one's own values. And sometimes, upon reflection, one may realize that some of the implicit values or premises are things they actually disagree with, once they are precisely spelled out.

In real-world issues, there are often many subtleties, and short arguments that sound airtight might be glossing over factors which are important in practice.

Example:
"During daylight, there is no need to have headlights (or running lights) on: there's already plenty of light for everybody to see each other by." Even during the day, headlights slightly increase how quickly other drivers see you during (say) a routine, tenth-of-a-second glance in their mirror.

Example:
"When in a turn-only lane, there is absolutely no need to signal — since there's only one way to turn, a signal can't communicating any information to other drivers!" Glib, but not true: Other defensive drivers presumably know you have only one legal option, but they don't know that **you** know that, and they are planning reactions in case you surprise them with a sudden illegal maneuver. By signaling, you give them information which helps them better plan for yet other contingencies. Furthermore, it also

gives you more confidence that other drivers are expecting your turn, reducing your suspicion that they're about to pull a surprise maneuver on you. (True, these are all low-probability events which almost always turn out to be unnecessary. But avoiding accidents is all about minimizing risks for the one moment events do spiral out of control.)

Example:
“You'll lose weight if and only if you burn more calories than you take in. All those diet-plan books can never get around this, and all their details are pointless.”
True, calorie intake and expenditure solely determine weight loss/gain. But after some thought, we can get examples where the above logic overlooks some relevant differences: If your friend told you they were switching from a diet of 2000 calories of balanced short-term and long-term energy sources (sugars, proteins, and carbs) to a diet of 2000 calories worth of Pixy Stix at breakfast plus a Flintstones multivitamin, would you be optimistic that they would have the willpower to strictly follow the new plan? The two plans are equal when counting calories, but in actuality one really is a better plan. (Even more exaggeratedly, consider a daily plan of 2000 calories of sugar while never drinking any water—since water has no calories, it can't affect your calorie count, according to the above claim.) These contrived counterexamples help illustrate that it's conceivable that there **can** be a difference between diet plans, so the initial claim isn't technically true.

The point illustrated is that often real-world arguments incorrectly imply that their result follows from the **form** of the argument, when in fact the form is not valid in the way a syllogism is. This fallacy can be illuminated by finding a different domain in which the argument fails. The practice of searching for domains which invalidate the argument can help both sides of a debate hone in on bringing the unspoken assumptions to light. The original argument, if its conclusion is indeed true, must be patched either by adding the unspoken assumptions or fixing the invalid form.

Exercise:

Problem:

Mistakes in syllogisms are hard to make: what are the only two ways to have an error in a syllogism?

Solution:

1. The argument isn't actually in syllogism form. For example, the following is an **incorrect** syllogism:

1	All people don't know my file's password.	Premise (Equivalent to “Nobody knows my file's password”, but reworded to be of the required form “All somethings have some property.”.)
2	All hackers are people.	Premise
3	Therefore, my file is secure from hackers.	**Incorrect** syllogism, lines 1,2

   To be a syllogism, the conclusion would have to be “all hackers don't know my file's password.” The file might or might not be secure, but the above doesn't prove it.
2. One of the two premises is wrong.

1	All people don't know my file's password.	Premise, **but possibly false**
2	All hackers are people.	Premise, **but possibly false**
3	Therefore, all hackers don't know my file's password.	Syllogism, lines 1,2

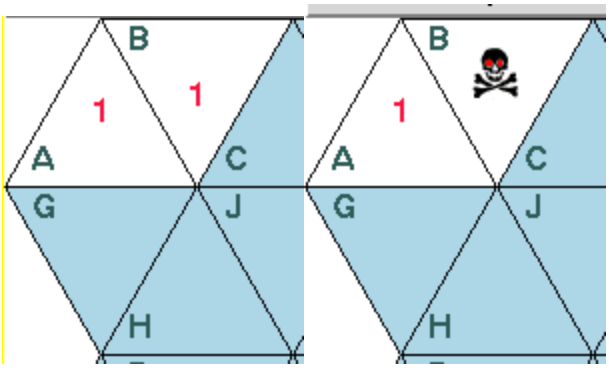
This proof fails, of course, if some hackers are non-people (e.g., programs), or if some people know the password. (In fact, presumably **you** know the password!)

Of course, even if a proof fails, the conclusion might be true for other reasons. An incorrect argument doesn't prove the conclusion's opposite!

Other Inference Rules

Of course, there are more ways to deduce things, beyond a syllogism.

- Who decides what the valid inference rules are?
- Is it always clear when people have used the inference rules correctly?



Glimpses of two different
WaterWorld boards

Consider the following argument about WaterWorld boards:

1	(A) is next to exactly onepirate.	Premise, from either subfigure
2	(A) has only one unexplored neighbor.	Premise, from either subfigure
3	If you are an unexpected location next to (A), then you contain apirate.	Incorrect conclusion

This conclusion is not valid; while it is correct for the first board shown, it is incorrect for the second. (I make this mistake all the time when playing WaterWorld too quickly, arrggh! — The Author.)

The problem is that the author of the argument presumably meant to conclude “all **explored** neighbors of (A) contain a pirate”.

Before we can study exact proofs, we need a way of writing exactly what we mean. This will occupy us for the next section.

The need for a precise language

These previous glitches in the WaterWorld arguments both arise, of course, because we were sloppy about what each sentence meant exactly. We used informal English — a fine language for humans, who can cope with

remarkable amounts of ambiguity -- but not a good language for specifying arguments.

Note: Laws and contracts are really written in a separate language from English — legalese — full of technical terms with specific meanings. This is done because, while some ambiguity is tolerable in 99% of human interaction, the remaining 1% can be very problematic. Even so, legalese still contains **intentionally** ambiguous terms: When, exactly, is a punishment "cruel and unusual"? What exactly is the "community standard" of indecency? The legal system tries to simultaneously be formal about laws, yet also be flexible to allow for unforeseen situations and situation-specific latitude. (The result of this tension is the position of Judge.)

Note: Court decisions, while dense reading, are often the model of well-presented arguments.

Consider, from a previous example, the statement "…[this is something] every Boy/Girl Scout and Architect should know". Does this mean all people who are **both** a scout and architect, or everybody who is at least one or the other? Genuinely ambiguous, in English! (Often, "and/or" is used to mean "one or the other or possibly both".)

We'll next look at a way to specify some concepts non-ambiguously, at least for WaterWorld. We need to be more careful about how we state our facts and how we use these known facts to deduce other facts. Remember, faulty reasoning might not just mean losing a silly game. Hardware and software bugs can lead to significant bodily harm (Imagine software bugs in an airplane autopilot or surgical robot system), security loopholes (e.g., in Mozilla or IE), or expensive recalls.

One reaction to the above arguments is “Well, big deal — somebody made a mistake (mis-interpreting or mis-stating a claim); that's their problem. (And sheesh, they sure are dolts!)” But as a programmer, that's not true: Writing large systems, human programmers **will** err, no matter how smart or careful or skilled they are. Type-checkers catch some errors upon compilation, and test suites catch their share of bugs, but many still remain in real-world software. Thus we are looking for systemic ways to reduce and catch errors, with the ultimate ideal of being able to prove programs correct.

Note: Other professions have checklists, protocols, and regulations to minimize human error; programming is no different, except that the industry is still working on exactly what the checklists or training should be. Someday, a license will be required for practicing software, at least for software involved with life-safety.

In our study of formal logic, we'll need three things:

- Syntax (language) — a precise syntax and vocabulary for expressing concepts without ambiguity,
    - Propositional logic,
    - First-order logic (propositional logic, plus relations and quantifiers)
- Semantics (meaning) and modeling — how to connect these formal languages to whatever topic we want to reason about (including our software).
- Reasoning (proofs) — methods of deducing new facts from old. We'll see three types of reasoning, and how to use them for each of our two logics:
    - Truth tables
    - Boolean Algebra

- Inference Rules

We'll visit these topics in an interleaved manner — first propositional logic (immediately with its semantics) and three methods of reasoning for it; then first-order logic and an in-depth look at its interpretations, and finally the methods of reasoning for first-order logic.

We'll begin with a particular syntax — propositional logic for the game of WaterWorld — before using this syntax to formally deduce safe moves.

propositions
(Blank Abstract)

Recall examples of where we'd like proofs:

- WaterWorld (Is a certain location guaranteed safe?)
- type checking (Does a program call functions in the proper way?)
- circuit verification (Does a circuit always work as advertised?)

After seeing the reasons why proofs are important, we ended with a call for first needing a precise language for writing down statements without the ambiguity of English.

Note: Might a programming language be a good way to specify formal concepts without ambiguity? Programming languages are usually motivated by specifying **how** to do something (implementation), rather than formally specifying **what** is being done (interface). While there is a deep relation between these two, logic is more appropriate for specifying the “what”.

A formal vocabulary

Imagine an offer where, for a mere $6.99, you can get: "EE, (FF or CF or OB or HB) or CC and PH and BR and GR or WB and PJ. Some fine print clarifies for us that BR includes T (Whi, Whe, Ra, or Hb), FT, HM (Bb, Ba, or Ca), EM, B with CrCh, BB (GR from 6-11am)." Unfortunately, it's not clear at all how the “and” and “or”s relate. Fundamentally, is “x and y or z” meant to be interpreted as “(x and y) or z ”, or as “x and (y or z)”? With some context, we might be able to divine what the author intended: the above offer is the direct translation from the menu of a local diner: "2 eggs, potatoes (french fries, cottage fries, O'Brien or hashed brown) or cottage cheese and peach half (grits before 11am) and choice of bread with gravy or whipped butter and premium jam. Bread choices include toast (white, wheat ,raisin or herb), hot flour tortillas,

homemade muffin (blueberry, banana or carrot), English muffin, bagel with cream cheese, homemade buttermilk biscuits. Grits available from 6:00am to 11:00am." (In a brazen display of understatement, this meal was called “Eggs Alone”.) Even given context, this offer still isn't necessarily clear to everybody: can I get both french fries and a peach half? Happily, coffee is available before having to decipher the menu. In this example, parentheses would have clarified how we should interpret “and”, “or”. But before we discuss how to connect statements, we will consider the statements themselves.

proposition
: A statement which can be either true or false.

Example:
“Your meal will include hashbrowns.”

propositional variable
: A variable that can either be true or false, representing whether a certain proposition is true or not.

Example:
HB

We will often refer to “propositional variables” as just plain ol' “propositions”, since our purpose in studying logic is to abstract away from individual statements and encapsulate them in a single variable, thereon only studying how to work with the variable.

For a proposition or propositional variable X, rather than write “X is true”, it is more succinct to simply write “X”. Likewise, “X is false” is indicated

as "$\neg X$".

Note: Compare this with Boolean variables in a programming language. Rather than `(x == true)` or `(x == false)`, it's idiomatic to instead write `x` or `!x`.

Observe that not all English sentences are propositions, since they aren't true/false issues. Which of the following do you think might qualify as propositions? If not, how might you phrase similar statements that are propositions?

- "Crocodiles are smaller than Alligators."
- "What time is it?"
- "Pass the salt, please."
- "Hopefully, the Rice Owls will win tomorrow's game."
- "Mr. Burns is filthy rich."
- "Fresca® is the bee's knees."

A particular vocabulary for WaterWorld

When playing WaterWorld, what particular propositions are involved? To consider this, we think of a generic board, and wonder what the underlying statements are. They are statements like "location A contains a pirate" ("$\mathrm{A\text{-}unsafe}$"), "location G has 2 adjacent pirates" ("$\mathrm{G\text{-}has\text{-}2}$") and so on. Each of these statements may be true or false, depending on the particular board in question.

Here are all the WaterWorld propositions that we'll use.

Remember that $\mathrm{B\text{-}unsafe}$ doesn't mean "I'm not sure whether or not B is safe"; rather it means "B is unsafe" — it contains a pirate. You may not be

sure whether (the truth of) this proposition follows what you see, but in any given board the variable has one of two values, true or false.

Every WaterWorld board has the same set of propositions to describe it: A-unsafe, B-has-2, etc. However, different boards will have different underlying values of those propositions.

Connectives

Some statements in the above proof were simple, e.g., the single proposition "A-has-2". Some statements had several parts, though, e.g., "(F-unsafe and G-unsafe)". We build these more complicated statements out of propositions. If you know both F-unsafe is false, and G-unsafe is false, what can you deduce about the truth of the statement "(F-unsafe and G-unsafe)"? Clearly, it is also false. What about when F-unsafe is false, but G-unsafe is true? What about when both propositions are true? In fact, we can fill in the following table:

a	b	$(a \wedge b)$
false	false	false
false	true	false
true	false	false
true	true	true

Truth table for ∧ (AND)

truth table

A truth table for an expression has a column for each of its propositional variables. It has a row for each different true/false combination of its propositional variables. It has one more column for the expression itself, showing the truth of the entire expression for that row.

Exercise:

Problem:

What do you think the truth table for "a or b" looks like? Hint: To fill out one row of the table, say, for $a = \text{true}$ and $b = \text{false}$, ask yourself "For this row, is it true that (a is true, or b is true)?"

Solution:

a	b	$(a \vee b)$
false	false	false
false	true	true
true	false	true
true	true	true

Truth table for $\vee$ (OR)

Exercise:

Problem:

The above proof also used subexpressions of the form "not b-unsafe". What is the truth table for "not a"?

Solution:

a	$\neg a$
false	true
true	false

Truth table for ¬ (NOT)

Exercise:

Problem: What is the truth table for the expression "(not a) or b"?

Solution:

a	b	$(a \Rightarrow b)$
false	false	true
false	true	true

---

a	b	$(a \Rightarrow b)$
true	false	false
true	true	true

Truth table for ⇒ (IMPLIES)

connective
: The syntactic operator combining one or more logical expressions into a larger expression.
: A function with one or more Boolean inputs and a Boolean result. I.e., the meaning of a syntactic operator.

Example:
Two operators are $\wedge$ and $\vee$.

Example:
The meaning of $\wedge$ and $\vee$, e.g., as described by their truth tables.

Example:
nand (mnemonic: “not and”), written $\uparrow$, takes in two Boolean values a and b, and returns true exactly when $a \ \wedge \ b$ is **not** true — that is, $a \uparrow b \equiv \neg (a \ \wedge \ b)$.

The following are the connectives we will use most often. At least some of these should already be familiar from Boolean conditional expressions.

Connective	Pronunciation	Meaning	Alternative pronunciations / notations
$\neg$	not	$\neg a$: a is false	-a; !a
$\wedge$	and	$a \ \wedge \ ba$ and b are both true	a*b; ab; a&&b; a&b
$\vee$	or	$a \ \vee \ b$: at least one of $\{a, b\}$ is true	a+b; a||b; a|b
$\Rightarrow$	implies	$a \Rightarrow b$: equivalent to $\neg a \ \vee \ b$	$a \rightarrow b$; $a \supset b$; if a then b; a only if b; b if a; b is necessary for a; a is sufficient for b

Connectives

Many other connectives can also be defined. In fact, it turns out that any connective for propositional logic can be defined in terms of those above.

Example:
Another connective is **if-and-only-if** or **iff**, written as $a \Leftrightarrow b$, which is true when a and b have the same truth value. So, as its name implies, it can be defined as $(a \Rightarrow b) \ \wedge \ (b \Rightarrow a)$. It is also commonly known as "a is equivalent to b" and "a is necessary and sufficient for b".

Exercise:

Problem:

Another connective is “exactly-one-of”, which is more traditionally called **exclusive-or** or **xor** (since it excludes both a and b holding, unlike the traditional “inclusive” or.) How would you define a “xor” b in terms of the above connectives?

Solution:

Exactly one is true if either (a is true, and b is false) or (a is false, and b is true). So, one way to define it is $a \oplus b \equiv a \ \wedge \ \neg b \ \vee \ \neg a \ \wedge \ b$.

The two halves of that formula also correspond to the two true rows of xor's truth table:

a	b	$(a \oplus b)$
false	false	false
false	true	true
true	false	true
true	true	false

Truth table for xor

Note that the conventional $a \ \vee \ b$ is sometimes called **inclusive-or**, to stress that it includes the case where both a and b hold.

In English, the word “or” may sometimes mean inclusive-or, and other times mean exclusive-or, depending on context. Sometimes the term “and/or” is used to emphasize that the inclusive-or really is intended.

Exercise:

Problem:

For each of the following English sentences, does “or” mean inclusive-or or exclusive-or?

1. “Whether you are tired or lazy, caffeine is just the drug for you!”
2. “Whether you win a dollar or lose a dollar, the difference in your net worth will be noticed.”
3. “If you own a house or a car, then you have to pay property tax.”
4. “Give me your lunch money, or you'll never see your precious hoppy taw again!”

Solution:

1. Inclusive.
2. Exclusive.
3. Inclusive.
4. Exclusive (hopefully).

formulas

Well-Formed Formulas

If we want to develop complicated expressions about breakfast foods like eggs, hashbrowns, and so on, we will want an exact grammar telling us how to connect the propositions, what connections are allowed, and when parentheses are necessary (if at all). We will choose a grammar so that all our formulas are fully parenthesized:

Well-Formed formula (WFF)

- A constant: true or false. (If you prefer brevity, you can write "T" or "F".)
- A propositional variable.
- A negation $\neg\varphi$, where φ is a WFF.
- A conjunction $\varphi \ \wedge \ \psi$, where φ and ψ are WFFs.
- A disjunction $\varphi \ \vee \ \psi$, where φ and ψ are WFFs.
- An implication $\varphi \Rightarrow \psi$, where φ and ψ are WFFs.

Example:
a

Example:
$\neg c$

Example:
$a \ \wedge \ \neg c$

Example:

$\neg c \ \vee \ a \ \wedge \ \neg c$, or equivalently, $(\neg c) \ \vee \ (a \ \wedge \ \neg c)$

Example:
$\neg c \ \vee \ a \ \wedge \ \neg c \Rightarrow b$, or equivalently, $((\neg c) \ \vee \ (a \ \wedge \ \neg c)) \Rightarrow b$

The last two examples illustrate that we can add parentheses to formulas to make the precedence explicit. While some parentheses may be unnecessary, over-parenthesizing often improves clarity. We introduced the basic connectives in the order of their precedence: ¬ has the highest precedence, while ⇒ has the lowest. Furthermore, ∧ and ∨ group left-to-right: $a \ \wedge \ b \ \wedge \ c \equiv (a \ \wedge \ b) \ \wedge \ c$, whereas ⇒ groups right-to-left.

Example:
We can combine these ways of forming WFFs in arbitrarily complex ways, for example,
$\neg \left((\neg a \ \wedge \ c \ \vee \ (b \Rightarrow a \Rightarrow c)) \ \wedge \ \neg (a \Rightarrow \neg b) \right)$
While **large** WFFs are common, and we will use some, ones with this much nesting are not.

Note: φ, ψ, and θ are **meta-variables** standing for any WFF. The literal character "φ" doesn't actually show up inside some WFF; but instead, any particular formula can be used where we write "φ". It is a variable which you the reader must substitute with some particular WFF, such as " $a \Rightarrow b$ ". Similarly, a, b, and c are meta-variables to be replaced with a proposition, such as "b".

Variations of well-formed formulas occur routinely in writing programs. While different languages might vary in details of what connectives are

allowed, how to express them, and whether or not all parentheses are required, all languages use WFFs.

Example:
When creating the homeworks' web pages, the authors keep the problems and solutions together in one file. Then, a program reads that file, and creates a new one which either excludes the solution (for the problem set), or includes it (for the solution set, and for practice-problems). The condition for deciding whether to include the solutions is a WFF.

```
;; is-a-solution?: paragraph -> boolean
;; A function to tell if we are looking at a
"solution" paragraph.
;; Assume this is provided.

;; is-in-a-practice-prob?: paragraph -> boolean
;; A function to tell if Is the current problem a
practice problem?
;; Assume this is provided.

;; include-all-solutions?: boolean
;; A variable for the entire file.
;; Assume this is provided.


;; show-or-hide-soln: paragraph -> paragraph
;; Either return the given paragraph,
;; or (if it shouldn't be revealed) return a
string saying so.
;;
(define (show-or-hide-soln a-para)
   (if (and (is-a-solution? a-para)
            (not (or include-all-solns? (is-in-a-
practice-prob? a-para)))
```

```
        "(see solution set)"
        a-para))
```

Note that the Boolean variable `include-all-solutions?` and Boolean values of `(is-a-solution? a-para)` and `(is-in-a-practice-prob? a-para)` play the part of propositions (is-soln, include-solns, is-practice), respectively. The `if`'s condition boils down to the WFF $\text{is-soln} \wedge \neg(\text{include-solns} \vee \text{is-practice})$.

Keep in mind that a WFF is purely a syntactic entity. We'll introduce rules later for re-writing or reasoning with WFFs, but it's those rules that will be contrived to preserve our meaning of connectives like ∧ or ¬. The truth value of a WFF depends on the truth values we assign to the propositions it involves.

When writing a program about WFFs, verifying syntactic property, calculating a value, counting the number of negations or bs, etc., such programs exactly follow the definition of WFF given.

Some formulas are truer than others

Is the formula $\text{A-unsafe} \vee \text{A-has-2}$ true? Your response should be that it depends on the particular board in question. But some formulas are true regardless of the board. For instance, $\text{A-unsafe} \vee \neg\text{A-unsafe}$: this holds no matter what. Similarly, $\text{A-unsafe} \wedge \neg\text{A-unsafe}$ can never be satisfied (made true), no matter how you try to set the variable A-unsafe.

truth assignment
: An assignment of a value true or false to each proposition being used.

Example:
For the formula $a \Rightarrow a \wedge b$, one possible truth assignment is $a = \text{true}$ and $b = \text{false}$. With that truth assignment, the formula is false.

Note:
We've used three different symbols to describe "equality" in some sense:

- $a \Leftrightarrow b$ is a formula. The symbol "⇔" is a logical connective.
- $\varphi \equiv \psi$ is a statement that two formulas are **equivalent** — that is, the same for all truth assignments.
- $a = \text{true}$ defines the value of a proposition. We also use the symbol for defining variables, $b = \psi$, and meta-variables, $\varphi = \psi$.

Of these, only "⇔" occurs **within** a formula.
Commonly, people use symbols such as "≡" for **multiple** purposes. This is problematic when part of what we are studying are the syntactic formulas themselves.

tautology
: A WFF which is true under any truth assignment (any way of assigning true/false to the propositions).

Example:
$\text{A-unsafe} \Rightarrow \text{A-unsafe}$

Example:
$a \Rightarrow a \ \vee \ b$

unsatisfiable
: A WFF which is false under any truth assignment.

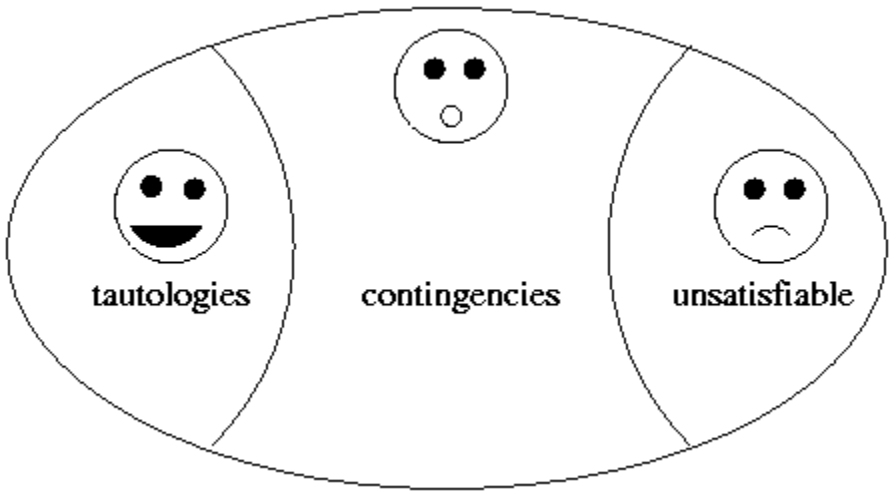
Example:

$\neg$ (A-unsafe $\Rightarrow$ A-unsafe)

Example:
$a \Rightarrow \neg a$

Note that in algebra, there are certainly formulas which are true (or similarly, false) for all values, but they don't get special names. For example, over the real numbers, **any** assignment to x makes the formula $x^2 \geq 0$ true, so it's similar to a tautology. Similarly, $x = x + 1$ is unsatisfiable, since it can't be made true for any assignment to x.

Some people use the term **contingency** to mean formulas in between: things which can be either true or false, depending on the truth assignment. Really, tautologies and unsatisfiable formulas are boring. However, trying to determine whether or not a formula is a tautology (or, unsatisfiable) **is** of interest. That's what proofs are all about!



Kinds of formulas: tautologies, contingencies, unsatisfiables

Identify the following Yogi Berra quotes either as tautologies, unsatisfiable, or neither. (Take these exercises with a grain of salt, since the English statements are open to some interpretation.)

Exercise:

Problem: " Pitching always beats batting — and vice-versa. "

Solution:

Unsatisfiable.

Exercise:

Problem: You can observe a lot just by watchin'.

Solution:

Tautology, arguably.

Exercise:

Problem: Nobody goes there anymore… it's too crowded.

Solution:

Unsatisfiable, unless of course you interpret "nobody" as "nobody of note".

Exercise:

Problem: It sure gets late early out here.

Solution:

Neither. If you interpret "gets late" as a social issue but "early" as a clock issue, then the statement might be true, depending on where

"here" is.

Exercise:

Problem:

Always go to other people's funerals; otherwise they won't come to yours.

Solution:

Unsatisfiable, except perhaps in a karmic sense.

Finding Truth

Now that we've seen how to express concepts as precise formulas; we would like to reason with them. By "reason", we mean some **automated** way of ascertaining or verifying statements — some procedure that can be carried out on an unthinking computer that can only push around symbols. In particular, for propositional logic, we'll restrict our attention to some (closely related) problems:

- **TAUTOLOGY**: given a formula φ, is it a tautology?
- **SAT**isfiability: Give a formula φ, is it satisfiable? (Is there some truth assignment to its variables, that makes it true?)
- **EQUIV**: Given two WFFs φ and ψ, are they equivalent? (Do they give the same result for all possible truth assignments to their variables?

Game-specific rules

Is $x \vee y \vee z$ a tautology? Clearly not. Setting the three propositions each to false, the formula is false. But now consider: Is $\text{A-has-0} \vee \text{A-has-1} \vee \text{A-has-2}$ a tautology? The answer here is " yes of course, … well, as long we're interpreting those propositions to refer to a WaterWorld board. " We'll capture this notion by listing a bunch of **domain axiom**s for WaterWorld: formulas which are true for all WaterWorld boards.

There are a myriad of domain axioms which express the rules of WaterWorld. Here are a few of them:

- $\text{A-has-0} \Rightarrow \text{B-safe} \ \wedge \ \text{G-safe}$
- $\text{A-has-2} \Rightarrow \text{B-unsafe} \ \wedge \ \text{G-unsafe}$
- $\ldots$

A more complete list is here. Whenever we deal with WaterWorld, we implicitly take all these domain axioms as given.

Propositional Logic: truth tables

Seeing how we can express some concepts as some formulas, and how some formulas are tautologies while others might be true or false depending on the truth assignment, we come to a question: **how** can we determine when a formula is a tautology? How can we tell if two different formulas are equivalent for all truth assignments? We'll look at three different methods of answering these questions:

- reasoning with truth tables (this section),
- reasoning with equivalences, and
- reasoning with inference rules.

Using Truth Tables

Is $\Rightarrow$ associative? In other words, is $a \Rightarrow (b \Rightarrow c)$ always equivalent to $a \Rightarrow b \Rightarrow c$? What is a **methodical** way of answering questions of this type? We can make a truth table with two output columns, one for each formula in question, and then just check whether those two columns are the same.
Exercise:

Problem: Use truth tables to show that $a \Rightarrow (b \Rightarrow c)$ and $(a \Rightarrow b) \Rightarrow c$ aren't equivalent.

Solution:

a	b	c	$(a \Rightarrow (b \Rightarrow c))$	$((a \Rightarrow b) \Rightarrow c)$
false	false	false	true	false
false	false	true	true	true
false	true	false	true	false
false	true	true	true	true
true	false	false	true	true
true	false	true	true	true
true	true	false	false	false
true	true	true	true	true

Truth table to check associativity of implication

By inspecting the two right-most columns, we see that the formulas are indeed not equivalent. They have different values for two truth-settings, those with $a = \text{false}$ and $c = \text{false}$.

Thus we see that truth tables are a method for answering questions of the form “Is formula φ equivalent to formula ψ?” We make a truth table, with a column for each of φ and ψ, and just inspect whether the two columns always agree. A bit of a brute-force solution, but certainly correct.

What about the related question, “Is formula θ a tautology?”. Well, obviously truth tables can handle this as well: make a truth table for the formula, and inspect whether all entries are `true`. For example, in the above problem, we could have made a truth table for the single formula $a \Rightarrow (b \Rightarrow c) \Leftrightarrow (a \Rightarrow b) \Rightarrow c$. The original question of equivalence becomes, is this new formula a tautology?

The first approach is probably a tad easier to do by hand, though clearly the two approaches are equivalent. Another handy trick is to have **three** output columns you're computing: one for $\varphi = a \Rightarrow (b \Rightarrow c)$, one for $\psi = (a \Rightarrow b) \Rightarrow c$, and one for $\varphi \Leftrightarrow \psi$; filling out the first two helper columns makes it easier to fill out the last column.

Note: When making a truth table for a large complicated WFF by hand, it's helpful to make columns for sub-WFFs; as you fill in a row, you can use the results of one column to help you calculate the entry for a later column.

Exercise:

Problem: Is it valid to replace the conditional

```
int do_something(int value1, int value2)
{
   bool a = …;
   bool b = …;

   if (a && b)
      return value1;
   else if (a || b)
      return value2;
   else
      return value1;
}
```

with this conditional?

```
int do_something(int value1, int value2)
{
   bool a = …;
   bool b = …;

   if ((a && !b) || (!a && b))
      return value2;
   else
      return value1;
}
```

After all, the latter seems easier to understand, since it has only two cases, instead of three.

Solution:

In the original code, we return `value2` when the first case is false, but the second case is true. Using a WFF, when $\neg (a \wedge b) \wedge (a \vee b)$. Is this equivalent to the WFF $a \wedge \neg b \vee \neg a \wedge b$? Here is a truth table:

a	b	$\neg (a \wedge b)$	$(a \vee b)$	$(\neg (a \wedge b) \wedge (a \vee b))$	$(a \wedge \neg b)$	$(\neg a \wedge b)$	$((a \wedge$
false	false	true	false	false	false	false	false
false	true	true	true	true	false	true	true
true	false	true	true	true	true	false	true
true	true	false	true	false	false	false	false

Truth table for comparing conditionals' equivalence

Yes, looking at the appropriate two columns we see they are equivalent.

So, how would do we use truth tables to reason about WaterWorld? Suppose you wanted to show that G-safe was true on some particular board. Clearly a truth table with the single column G-safe alone isn't enough (it would have only two rows — false and true — and just sit there and stare at you). We need some way to incorporate both the rules of WaterWorld and the parts of the board that we could see.

We can do that by starting with a **huge** formula that was the conjunction of all the WaterWorld domain axioms; call it ρ. We would encode the board's observed state with another formula, ψ. Using these, we can create the (rather unwieldy) formula that we're interested in: $\rho \wedge \psi \Rightarrow$ G-safe. (Notice how this formula effectively ignores all the rows of the the truth-table that don't satisfy the rules ρ, and the rows that don't correspond to the board we see ψ: because of the semantics of $\Rightarrow$, whenever $\rho \wedge \psi$ is false, the overall formula $\rho \wedge \psi \Rightarrow$ G-safe is true.)

Propositional Logic: limitations of truth tables

Are we done yet?

Are we done with propositional logic, now that we can test for equivalence and tautologies, using truth tables? Possibly. Truth tables can answer any question about propositional logic, but not always conveniently.

Consider the following code:

```
bool do_something(int value)
{
   bool a = …;
   bool b = …;

   if (a && !b)
      return true;
   else if (!a && !b)
      return false;
   else if (a)
      return a;
   else if (b)
      return false;
   else
      return true;
}
```

Clearly, this is very ugly and should be simplified. But to what? We could build a truth table for the corresponding WFF, but so far we don't have any better way of finding a simpler equivalent formula than testing equivalence with whatever comes to mind. We need some way to generate formulas, given either an equivalent formula or a truth table.

There is another practical difficulty with truth table: they can get unwieldy.
Exercise:

Problem:

How many rows are there in a truth table with 2 input variables? 3 variables? 5 variables? 10 variables? n variables?

Solution:

- 2 variables: As we're seen, 4 rows.
- 3 variables: 8 rows.
- 5 variables: 32 rows.
- 10 variables: 1024 rows.
- n variables: 2^n rows.

Exercise:

Problem:

(**Optional**) Now, how many such boolean **functions** are possible, with 2 inputs? With 3? For fun, sit down and name all the possible two-input functions. You'll find that some of them are rather boring, such as the constant function $\mathtt{true}$, and many are just permutations on $\Rightarrow$.

Solution:

- With 2 variables, we have 4 rows. How many different ways can we assign $\mathtt{true}$ and $\mathtt{false}$ to those 4 positions? If you write them all out, you should get 16 combinations.
- With 3 variables, we have 8 rows and a total of 256 different functions.
- With n variables, we have 2^n rows and a total of 2^{2^n} different functions. That's a lot!

When discussing a circuit with 100 wires (each corresponding to a single proposition), truth tables are clearly infeasible. Modern processors have **millions** of wires and transistors. It is still an area of active research to cope with such a huge number of possibilities. (The key idea is to break things

down into small sections, prove things about the small sections, and hopefully have a small set of sentences formally capturing the interface between sections.)

So truth tables are intractable for analyzing circuits of more than a few wires. But will they suffice for answering WaterWorld questions? Image a (large) table with all the neighbor propositions: A-has-0, B-has-0, …, A-has-1, B-has-1, … Now, determine which rows which entail B-safe. To answer this, we end up looking at rows involving many clearly-irrelevant propositions such as Z-has-2.

Note: Hmm, considering every possible board and then counting what proportion of boards entail B-safe — hmm, this is the brute-force definition of probability! Since such truth tables enumerates all possible boards, it's like looking for probability 1 the brute-force way.

Also, this method of playing WaterWorld via huge truth tables would be unsatisfying for another reason: it doesn't actually reflect our own reasoning. As a general principle of programming, your program should always reflect how **you** conceive of the problem. The same applies to logic.

Note: Consider the difference between using truth tables and actually reasoning. The philosopher Bertrand Russell, trying to pin down what exactly constitutes “knowledge”, suggested that he knows that the last name of Britain's prime minister begins with a 'B'. While Gordon Brown is prime minister, making Bertrand is correct, we hesitate to say he actually **knows** the fact — he wrote his example when the prime minister was Arthur Balfour (1902-1905). So while he is correct in a truth-table sense, his reasoning isn't, and we tend to say that he does not actually **know** the prime minister's last initial.

So, no: we're not yet finished with propositional logic. We want to look for (hopefully) more feasible ways to determine whether a formula is a tautology (or, whether two formulas are equivalent). As a clue, we'll try to discover methods which are based on the way we naively approach this. We'll look first at equivalences, and then at inference rules.

Propositional Logic: equivalences

Propositional Equivalences

What are the roots of $x^3 - 4x$? Well, in high-school algebra you learned how to deal with such numeric formulas:

	$x^3 - 4x$	
$=$	$x\left(x^2 - 4\right)$	factor out x
$=$	$x\left(x - 2\right)\left(x + 2\right)$	The identity $a^2 - b^2 = (a + b)\,(a - b)$ with a being x, and b being 2.

This last expression happens to be useful since it is in a form which lets us read off the roots 0, +2, -2. The rules of algebra tell us that these three **different** formulas are all equivalent. In fact, our very definition of two formulas being **equivalent** is that for any value of x the two formulas return the same value. We are distinguishing between **syntax** (the expression itself, as data), and **semantics** (what the expression means). Usually, when presented with syntax, one is supposed to bypass that and focus on its meaning (e.g., reading a textbook). However, in logic and post-modern literature alike, we are actually studying the interplay between syntax and semantics. The general gist is that in each step, you rewrite subparts of your formula according to certain rules ("replacing equals with equals").

Well, we can use a similar set of rules about rewriting formulas with equivalent ones, to answer the questions of whether two formulas are equal, or whether a formula is a tautology. George Boole was the first to realize that $\mathtt{true}$ and $\mathtt{false}$ are just values in the way that numbers are, and he first

codified the rules for manipulating them; thus **Boolean algebra** is named in his honor.

Note: The term “algebra” comes from the values true, false and operators $\wedge$, $\vee$ having some very specific properties similar to those of numbers with $\times$, $+$.

George Boole (1815-1864)

Again, each individual step consists of rewriting a formula according to certain rules. So, just what are the rules for manipulating Boolean values? We'll start with an example.

Example:

1	$a \wedge \text{false} \vee b \wedge \text{true}$	
2	$\equiv \text{false} \vee b \wedge \text{true}$	Dominance of false over $\wedge$
3	$\equiv b \wedge \text{true} \vee \text{false}$	Commutativity of $\vee$
4	$\equiv b \wedge \text{true}$	Identity element for $\vee$ is false
5	$\equiv b$	Identity element for $\wedge$ is true

Thus we have a series of equivalent formulas, with each step justified by citing a propositional equivalence. By and large, the equivalences are rather mundane. A couple are surprisingly handy; take a moment to consider **DeMorgan's laws**.

$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$	$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$

(Try φ being “Leprechauns are green”, and ψ being “Morgana Le Fay likes gold”. Do these laws make sense, for each of the four possible truth assignments?) Augustus DeMorgan was also an important figure in the formalization of logic.

Augustus DeMorgan (1806-1871)

Here is another example. For a statement $\varphi \Rightarrow \psi$, the **contrapositive** of that formula is $\neg\psi \Rightarrow \neg\varphi$. We can show that a formula is equivalent to its contrapositive:

Example:
Contrapositive

1	$\varphi \Rightarrow \psi$	

2	$\equiv \neg\varphi \ \vee \ \psi$	Definition of $\Rightarrow$
3	$\equiv \psi \ \vee \ \neg\varphi$	Commutativity of $\vee$
4	$\equiv \neg\neg\psi \ \vee \ \neg\varphi$	Double Complementation
5	$\equiv \neg\psi \Rightarrow \neg\varphi$	Definition of $\Rightarrow$

Don't confuse the contrapositive of a statement with the **converse** of a formula: The converse of $\varphi \Rightarrow \psi$ is the formula $\psi \Rightarrow \varphi$; in general **a formula is not equivalent to its converse**!

This next example is actually a proof of one of the laws from the given list, using (only) others from the list.

Example:
Absorption of $\vee$

1	$\varphi \ \wedge \ \psi \ \vee \ \psi$	
2	$\equiv \varphi \ \wedge \ \psi \ \vee \ \psi \ \wedge \ \mathrm{true}$	Identity of $\wedge$
3	$\equiv \psi \ \wedge \ \varphi \ \vee \ \psi \ \wedge \ \mathrm{true}$	Commutativity of $\vee$
4	$\equiv \psi \ \wedge \ (\varphi \ \vee \ \mathrm{true})$	Distributivity of $\wedge$ over $\vee$
5	$\equiv \psi \ \wedge \ \mathrm{true}$	Dominance of $\vee$

6	$\equiv \psi$	Identity of $\wedge$

Exercise:

Problem:

Show that the "Absorption of $\wedge$" equivalence holds, given the other equivalences. I.e., show $(a \ \vee \ b) \ \wedge \ b \equiv b$.

Solution:

1	$(a \ \vee \ b) \ \wedge \ b$	
2	$\equiv (a \ \vee \ b) \ \wedge \ (b \ \vee \ \text{false})$	Identity of $\vee$
3	$\equiv (b \ \vee \ a) \ \wedge \ (b \ \vee \ \text{false})$	Commutativity of $\vee$
4	$\equiv b \ \vee \ a \ \wedge \ \text{false}$	Distributivity of $\vee$ over $\wedge$
5	$\equiv b \ \vee \ \text{false}$	Dominance of $\wedge$
6	$\equiv b$	Identity of $\vee$

Compared to proofs using truth tables, Boolean algebra gives us much shorter proofs. But, determining **which** equivalence to use in the next step of a proof can be difficult. In this case, compare the solution for this

exercise to the previous absorption proof. These two proofs have a special **dual** relationship described in the next section.

Exercise:

Problem:

Show that the **modus ponens** rule, $a \ \wedge \ (a \Rightarrow b) \Rightarrow b$ always holds. I.e., show that it is a tautology, and thus equivalent to true.

Solution:

1	$a \ \wedge \ (a \Rightarrow b) \Rightarrow b$	
2	$\equiv a \ \wedge \ (\neg a \ \vee \ b) \Rightarrow b$	Definition of $\Rightarrow$
3	$\equiv a \ \wedge \ \neg a \ \vee \ a \ \wedge \ b \Rightarrow b$	Distributivity of $\vee$ over $\wedge$
4	$\equiv \mathrm{false} \ \vee \ a \ \wedge \ b \Rightarrow b$	Complement
5	$\equiv a \ \wedge \ b \ \vee \ \mathrm{false} \Rightarrow b$	Commutativity of $\vee$
6	$\equiv a \ \wedge \ b \Rightarrow b$	Identity of $\vee$
7	$\equiv \neg \left(a \ \wedge \ b\right) \ \vee \ b$	Definition of $\Rightarrow$
8	$\equiv \neg a \ \vee \ \neg b \ \vee \ b$	DeMorgan's law
9	$\equiv \neg a \ \vee \ \neg b \ \vee \ b$	Associativity of $\vee$
10	$\equiv \neg a \ \vee \ b \ \vee \ \neg b$	Commutativity of $\vee$
11	$\equiv \neg a \ \vee \ \mathrm{true}$	Complement

12	$\equiv \text{true}$	Dominance of $\vee$

So, what would it mean to use Boolean algebra as reasoning for WaterWorld? That is, if you wanted to show that G-safe was true, how would you do that using Boolean algebra? As with truth-tables, we would take the conjunction of all the WaterWorld domain axioms (call it ρ), **and** the board's observed state (ψ). We would then want to show that asserting G-safe was already equivalent to the rules-and-observed-state: $\rho \wedge \psi \equiv \rho \wedge \psi \wedge \text{G-safe}$.

Duals (optional)

Duals: a symmetry between $\wedge$, $\vee$ mediated by $\neg$.

Looking at the provided propositional equivalences, you should notice a strong similarity between those for $\vee$ and those for $\wedge$. Take any equivalence, swap $\vee$s and $\wedge$s, swap trues and falses, and you'll have another equivalence! For instance, there are two flavors of DeMorgan's law, which are just duals of each other:

$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$	$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$

Note: In terms of circuit diagrams, we can change each AND gate to an OR gate and add negation-bubbles to each gate's inputs and outputs. The principle of duality asserts that this operation yields an equivalent circuit.

The idea of duality is more general than this. For example, polyhedra have a natural dual of interchanging the role of vertices and faces.

Propositional Logic: normal forms

CNF, DNF, … (ENufF already!)

In high school algebra, you saw that while $x^3 - 4x$ and $x\,(x-2)\,(x+2)$ are equivalent, the second form is particularly useful in letting you quickly know the roots of the equation. Similarly, in Boolean algebra there are certain canonical — "normal" — forms which have nice properties.

A formula in **Conjunctive Normal Form**, or **CNF**, is the conjunction of **CNF clauses**. Each clause is a formula of a simple form: a disjunction of possibly-negated propositions.

Example:
$c \Rightarrow a \ \wedge \ b$ is equivalent to $(a \ \vee \ \neg c) \ \wedge \ (b \ \vee \ \neg c)$. This latter formula is in CNF, since it is the conjunction of disjunctions, and each disjunction consists only of propositions and negated propositions.

Example:
The conjunctions and disjunctions need not be binary. The following formula is also is CNF.
$\neg a \ \wedge \ (a \ \vee \ b \ \vee \ \neg c) \ \wedge \ (b \ \vee \ \neg d \ \vee \ e \ \vee \ f)$
Note that its first clause is just one negated proposition. It is still appropriate to think of this as a disjunction, since $\varphi \equiv \varphi \ \vee \ \varphi$.

Another format, **Disjunctive Normal Form**, or **DNF** is the dual of conjunctive normal form. A DNF formula is the disjunction of **DNF clauses**, each a conjunction of possibly-negated propositions.

Example:
$a \ \wedge \ b \Rightarrow c$ is equivalent to $\neg a \ \vee \ \neg b \ \vee \ c$ which is in DNF: three disjunctions, each being a clause with only one term. (It also happens to be in CNF — a single clause with three terms!) It is also equivalent to the more fleshed out DNF formula where we insist that each clause include all three variables. We end up with a formula that includes each possible clause **except** $a \ \wedge \ b \ \wedge \ \neg c$: That is, the formula
$(a \ \wedge \ b \ \wedge \ c) \ \vee \ (a \ \wedge \ \neg b \ \wedge \ c) \ \vee \ (a \ \wedge \ \neg b \ \wedge \ \neg c) \ \vee \ (\neg a \ \wedge \ b \ \wedge \ c) \ \vee \ (\neg a \ \wedge \ b \ \wedge \ \neg c) \ \vee \ (\neg a \ \wedge \ \neg$
.

Note: Electrical Engineering courses, coming from more of a circuit perspective, sometimes call CNF **product-of-sums**, and call DNF **sum-of-products**, based on $\vee$,$\wedge$ being analogous to +,*.

Any Boolean function can be represented in CNF and in DNF. One way to obtain CNF and DNF formulas is based upon the truth table for the function.

- A DNF formula results from looking at a truth table, and focusing on the rows where the function is true: As if saying "I'm in this row, or in this row, or …": For each row where the function is true, form a conjunction of the propositions. (E.g., for the row where a is false, and b is true, form $\neg a \ \wedge \ b$.) Now, form the disjunction of all those conjunctions.
- A CNF formula is the pessimistic approach, focusing on the rows where the function is false: "I'm not in this row, and not in this row, and …". For each row where the function is false, create a formula for "not in this

row": (E.g., if in this row a is false and b is true form $\neg\left(\neg a \ \wedge \ b\right)$; then notice that by DeMorgan's law, this is $a \ \vee \ \neg b$ — a disjunct. Now, form the conjunction of all those disjunctions.

Example:

a	b	c	**Unknown function**
false	false	false	false
false	false	true	false
false	true	false	true
false	true	true	true
true	false	false	false
true	false	true	true
true	true	false	false
true	true	true	false

Truth table example

For CNF, the false rows give us the following five clauses:

- $a \ \vee \ b \ \vee \ c$
- $a \ \vee \ b \ \vee \ \neg c$
- $\neg a \ \vee \ b \ \vee \ c$
- $\neg a \ \vee \ \neg b \ \vee \ c$
- $\neg a \ \vee \ \neg b \ \vee \ \neg c$

and the full formula is the conjunction of these. Essentially, each clause rules out one row as being true.
For DNF, the true rows give us the following three clauses:

- $\neg a \ \wedge \ b \ \wedge \ \neg c$
- $\neg a \ \wedge \ b \ \wedge \ c$
- $a \ \wedge \ \neg b \ \wedge \ c$

and the full formula is the disjunction of these. Essentially, each clause allows one row to be true.

This shows that, for any arbitrarily complicated WFF, we can find an equivalent WFF in CNF or DNF. These provide us with two very regular and relatively uncomplicated forms to use.
Exercise:

Problem:

The above example produced CNF and DNF formulas for a Boolean function, but they are **not** the simplest such formulas. For fun, can you find simpler ones?

Solution:

- CNF: $(a \vee b) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b)$
- DNF: $(\neg a \wedge b) \vee (a \wedge \neg b \wedge c)$

Note: Karnaugh maps are a general technique for finding minimal CNF and DNF formulas. They are most easily used when only a small number of variables are involved. We won't worry about minimizing formulas ourselves, though.

Notation for DNF, CNF

Sometimes you'll see the form of CNF and DNF expressed in a notation with subscripts.

- DNF is $\vee_i \psi_i$, where each clause φ_i is $\wedge_j \lambda_j$, where each λ is a propositional variable (Prop), or a negation of one ($\neg\,\mathrm{Prop}$).
- CNF is $\wedge_i \psi_i$, where each clause ψ_i is $\vee_j \lambda_j$, where each λ is again a propositional variable (Prop), or a negation of one ($\neg\,\mathrm{Prop}$).

For example, in the CNF formula $(a \vee b) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b)$ we have $\varphi_2 = \neg a \vee b \vee c$ within that clause we have $\lambda_1 = \neg a$.

One question this notation brings up:

- What is the disjunction of a single clause? Well, it's reasonable to say that $\psi \equiv \psi$. Note that this is also equivalent to $\psi \vee \mathrm{false}$.
- What is the disjunction of zero clauses? Well, if we start with $\psi \equiv \psi \vee \mathrm{false}$ and remove the ψ, that leaves us with false! Alternately, imagine writing a function which takes a list of booleans, and returns the $\vee$ of all of them — the natural base case for this recursive list-processing program turns out to be false. Indeed, this is the accepted definition of the empty disjunction. It follows from false being the identity element for $\vee$. Correspondingly, a conjunction of zero clauses is true.

Actually, that subscript notation above isn't **quite** correct: it forces each clause to be the same length, which isn't actually required for CNF or DNF. For fun, you can think about how to patch it up. (Hint: double-subscripting.)

Note that often one of these forms might be more concise than the other. Here are two equivalently verbose ways of encoding true, in CNF and DNF respectively: $(a \vee \neg a) \wedge (b \vee \neg b) \wedge \ldots \wedge (z \vee \neg z)$ is equivalent to $(a \wedge b \wedge c \wedge \ldots \wedge y \wedge z) \vee (a \wedge b \wedge c \wedge \ldots \wedge y \wedge \neg z) \vee (a \wedge b \wedge c \wedge \ldots \wedge \neg y \wedge z) \vee$. The first version corresponds to enumerating the choices for each location of a WaterWorld board; it has 26 two-variable clauses. This may seem like a lot, but compare it to the second version, which corresponds to enumerating all possible WaterWorld boards explicitly: it has all possible 26-variable clauses; there are $2^{26} \approx 64$ billion of them!

soundness and completeness
The soundness and completeness of a system.

Are we done yet?

We have shown procedures, using both truth tables and equivalences, for solving two different logic problems:

- **Equivalence**: Show whether or not two WFFs φ and ψ are equivalent (the same under any truth assignment);
- **Tautology**: Show whether or not a given WFF φ is a tautology (true under all truth assignments).

Exercise:

Problem:

Which of these two logic problems seems harder than the other? That is, suppose you have a friend who can solve **any** Equivalence problem efficiently. But you want to open a business which will solve **any** Tautology problem efficiently. Can you open your business and, by subcontracting out specific Equivalence problems to your friend, really solve any Tautology problem brought to you? This question is sometimes phrased as " Does Tautology **reduce** to Equivalence? " Or, does it work the other way: does Equivalence reduce to Tautology?

Solution:

We can indeed reduce the question of Tautology to the question of Equivalence: if somebody asks you whether φ is true, you can just turn around and ask your friend whether the following two formulas are equivalent: φ, and true. Your friend's answer for this variant question will be your answer to your customer's question about φ. Thus, the Tautology problem isn't particularly harder than the Equivalence problem.

But also, Equivalence can be reduced to Tautology: if somebody asks you whether φ is equivalent to ψ, you can construct a new formula

$(\varphi \Rightarrow \psi) \ \wedge \ (\psi \Rightarrow \varphi)$. This formula is true exactly when φ and ψ are equivalent. So, you ask your friend whether this bigger formula is a tautology, and you then have your answer to whether the two original formulas were equivalent. Thus, the Equivalence problem isn't particularly harder than the Tautology problem!

Given these two facts (that each problem reduces to the other), we realize that really they are essentially the same problem, in disguise.

But we have a more fundamental question to ask, about the method of using Boolean algebra (propositional equivalences) to prove something: Where does the initial list of allowable equivalences come from, and how do we know they're valid? The answer is easy — each equivalence can be verified by a truth table!

Exercise:

Problem:

Using a truth table, show the validity of conjunctive Redundancy:
$\varphi \ \wedge \ (\neg\varphi \ \vee \ \psi) \equiv \varphi \ \wedge \ \psi$

Solution:

Compare the last two columns in the following:

a	b	$\neg a \ \vee \ b$	$a \ \wedge \ (\neg a \ \vee \ b)$	$a \ \wedge \ b$
false	false	true	false	false
false	true	true	false	false
true	false	false	false	false

a	b	$\neg a \lor b$	$a \land (\neg a \lor b)$	$a \land b$
true	true	true	true	true

Truth table to prove validity of conjunctive Redundancy

This is called **soundness** of Boolean algebra: If, using our propositional equivalence rules, we derive that φ and ψ are equivalent, then truly they are equivalent. (Whew!)

By the way, there is one subtle point: our truth table tells us that $a \land b$ and $b \land a$ are equivalent. But then suddenly we generalize this to saying that for **any** formulas φ and ψ, $\varphi \land \psi$ and $\psi \land \varphi$ are also equivalent. What lets us justify that step? It's because any given formula will be either true or false, so we can reduce the entire formula to a single true/false proposition.

Is Boolean algebra enough? Does our list of allowable propositional equivalences include everything you'll need? That is, could I have asked as a homework problem to show some two formulas equivalent (using Boolean algebra), and even though they really are equivalent, there aren't enough rules to on our list to let you finish the homework? Hmm, good question! The property we desire here is called the **completeness** of Boolean algebra: any equivalence which is true can be proved.

It turns out that, given any two formulas which really are equivalent, Boolean algebra **is** indeed sufficiently powerful to show that. Put both formulas into CNF (or, DNF); if the truth tables are equal then the CNF formulas will be equal. (Well, there are a few details to take care of: you have to order the clauses alphabetically, eliminate any duplicate clauses, and include all variables in each clause. This might be tedious, but not difficult.) Thus, Boolean algebra is complete, since (we state without proof) this procedure can always be carried out.

The concepts of soundness and completeness can be generalized to any system.

soundness
: If the system (claims to) prove something is true, it really is true.

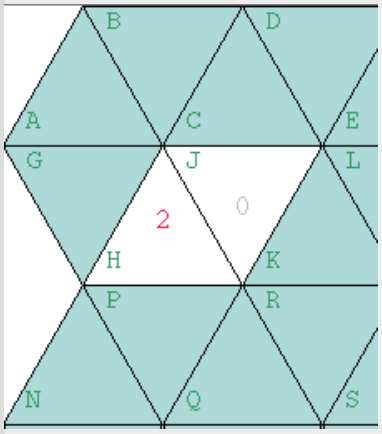
completeness
: If something really is true, the system is capable of proving it.

Propositional Logic: inference rules

Inference

Truth tables and equivalences are useful and powerful tools, but they do not correspond to how we usually reason about things. What we will do now is look at more familiar reasoning and how to formalize that. For example, with Boolean algebra it is awkward to prove that $a \wedge b$ implies a. For that, it is necessary to reword the problem in terms of equivalences, as $a \wedge b \Rightarrow a \equiv \text{true}$. Our next tool provides a more straightforward way to reason about implications.

Example:
Given the following piece of a WaterWorld board, how would you conclude that G is unsafe?



A glimpse of a WaterWorld board

Since H-has-2, at least two of H's three neighbors must be unsafe. But, since we know that one of these, J, isn't unsafe, then the two others, including G, must both be unsafe. Let's write this out more explicitly:

1	H-has-2would imply one of the following is true: (P-unsafe and G-unsafe), or (J-unsafe and P-unsafe), or (G-unsafe and J-unsafe).	WaterWorld domain axiom, i.e., definition of H-has-2

2	H-has-2 is true.	Premise (by inspection of this particular board)
3	One of the following is true: (P-unsafe and G-unsafe), or (J-unsafe and P-unsafe), or (G-unsafe and J-unsafe).	lines 1,2
4	not J-unsafe	Premise (by inspection)
5	(P-unsafe and G-unsafe)	lines 3,4
6	G-unsafe	line 5

Whew! A lot of small steps are involved in even this small deduction. It's apparent we'd want to automate this as much as possible! Let's look at some other short examples, which we'll formalize in a moment.

Exercise:

Problem: How do you know that A-has-2 proves B-unsafe ?

Solution:

Intuitively, this is straightforward. Since A-has-2, then both of its two neighbors, including B, must be unsafe. For this problem, let's be a bit more formal and use WFFs instead of prose in the steps.

1	A-has-2	Premise
2	A-has-2 $\Rightarrow$ B-unsafe $\wedge$ F-unsafe	WaterWorld domain axiom, i.e., definition of A-has-2
3	B-unsafe $\wedge$ F-unsafe	lines 1,2
4	B-unsafe	line 3

Exercise:

Problem: Similarly, how do you reason that A-has-1 and G-safe prove B-unsafe?

Solution:

Again a similar idea, if A-has-1, then at least one of A's two neighbors must be unsafe. But, since we know that one of these, G isn't unsafe, then the other, B, must be unsafe.

1	$\text{A-has-1} \Rightarrow \text{B-safe} \ \wedge \ \text{G-unsafe} \ \vee \ \text{B-unsafe} \ \wedge \ \text{G-safe}$	WaterWorld domain axiom
2	A-has-1	Premise
3	$\text{B-safe} \ \wedge \ \text{G-unsafe} \ \vee \ \text{B-unsafe} \ \wedge \ \text{G-safe}$	lines 1,2
4	G-safe	Premise
5	$\text{B-unsafe} \ \wedge \ \text{G-safe}$	lines 3,4
6	B-unsafe	line 5

Formal inference rules and proofs

In the above examples, we relied on common sense to know what new true formulas could be derived from previous ones. Unfortunately, common sense is imprecise and sometimes wrong. So, we need to formalize how we form proofs.

We now define a **formal proof** of θ from the **premises** φ, …, ψ, written
Equation:

$$\varphi,\ldots,\psi \vdash \theta$$

(A proof with no premises simply means there is nothing on the left of the turnstile: $\vdash \theta$.) For example, we'll show shortly that $\text{H-has-2} \vdash \text{G-unsafe}$. A proof consists of a sequence of WFFs, each with a justification for its truth. We will describe four permissible justifications for each step:

- A premise.
- An axiom.
- An inference rule.
- A subproof.

Note: Officially we might want to annotate the turnstile with "ww", to mean "proves within the WaterWorld inference system", indicating our use of the WaterWorld domain axioms. If you're proving things about other domains, you'd use different domain axioms.

Example:
We can formalize the above examples to show each of the following:

- $\text{H-has-2} \vdash \text{G-unsafe}$
- $\text{A-has-2} \vdash \text{B-unsafe}$
- $\text{A-has-1}, \text{G-safe} \vdash \text{B-unsafe}$

See below for formal proofs of some of these.

Stating an **axiom**, a simple assumed truth, is a rather trivial, boring way of coming up with a true formula. Some axioms are **domain axioms**: they pertain only to the domain you are considering, such as WaterWorld. In our case, we don't have any axioms that aren't domain axioms. If our domain were arithmetic, our axioms would describe how multiplication distributes over addition, etc.

Just using axioms is not enough, however. The interesting part is to deduce new true formulas from axioms and the results of previous deductions.

Note: "The point of philosophy is to start with something so simple as not to seem worth stating, and to end with something so paradoxical that no one will believe it." *Bertrand Russell, The Philosophy of Logical Atomism*

An **inference rule** formalizes what steps are allowed in proofs. We'll use this list of valid inference rules as our definition, but, this is just one set of possible inference rules, and other people could use slightly different ones.

First, let's look at some simple examples, using the simpler inference rules.

Example:
We'll formalize a previous exercise to show $\text{A-has-2} \vdash \text{B-unsafe}$.

1	A-has-2	Premise
2	$\text{A-has-2} \Rightarrow \text{B-unsafe} \ \wedge \ \text{F-unsafe}$	WaterWorld domain axiom
3	$\text{B-unsafe} \ \wedge \ \text{F-unsafe}$	$\Rightarrow$Elim, lines 1,2, where $\varphi = \text{A-has-2}$, and $\psi = \text{B-unsafe} \ \wedge \ \text{F-unsafe}$
4	B-unsafe	$\wedge$Elim (left), line 3, where $\varphi = \text{B-unsafe}$, and $\psi = \text{F-unsafe}$

What we mean in line 3, for example, is that we are using the domain axiom $\Rightarrow$Elim. That states that if we know $\varphi \Rightarrow \psi$, and we know φ, then we can conclude ψ. In line 3, we have defined $\varphi = \text{A-has-2}$ and $\psi = \text{B-unsafe} \ \wedge \ \text{F-unsafe}$, so that $\varphi \Rightarrow \psi$ corresponds to the conclusion of line 2 and φ corresponds to that of line 1. Thus, this domain axiom applies, and we get the conclusion ψ.
That's almost exactly like the steps we took in the previous informal proof, but now we're a bit pickier about our justifications for each step.

Formally, when using a domain axiom, the justification is a combination of the name of that inference rule, the line numbers of which previous WFFs are being used, and a description of how those WFFs are used in that inference rule in this particular step. Later, we'll often omit the description of exactly how the specific inference rule is used, since in many cases, that information is painfully obvious.

Example:
In this system, commutativity of $\wedge$ and $\vee$ are not among the inference rules. However, they do follow. For example, consider the following proof of $A \wedge B \vdash B \wedge A$.

1	$A \wedge B$	Premise
2	A	$\wedge$Elim (left), line 1, where $\varphi = A$
3	B	$\wedge$Elim (right), line 1, where $\psi = B$
4	$B \wedge A$	$\wedge$Intro, lines 3,2, where $\varphi = B$, and $\psi = A$

Does this example also show that $C \wedge D \vdash D \wedge C$? Well, yes and no. That proof does **not** have anything to do with propositions C and D. But, clearly, we could create another nearly identical proof for $C \wedge D \vdash D \wedge C$, by substituting C and D for A and B, respectively. What about proving the other direction of commutativity: $B \wedge A \vdash A \wedge B$? Once again, the proof has exactly the same form, but substituting B and A for A and B, respectively. Stating such similar proofs over and over is technically necessary, but not very interesting. Instead, when the proof depends solely on the form of the formula and not on any axioms, we'll use meta-variables to generalize.

Example:
Generalized $\wedge$ commutativity: $\chi \wedge \upsilon \vdash \upsilon \wedge \chi$

1	$\chi \wedge \upsilon$	Premise
2	υ	$\wedge$Elim (left), line 1, where $\varphi = \chi$
3	χ	$\wedge$Elim (right), line 1, where $\psi = \upsilon$

4	$\upsilon \wedge \chi$	$\wedge$Intro, lines 3,2, where $\varphi = \upsilon$, and $\psi = \chi$

Exercise:

Problem:

Similarly, associativity of $\wedge$ and $\vee$ are not among the inference rules. This is a particularly important detail, since our WaterWorld domain axioms frequently use formulas of the form $a \wedge b \wedge c$, which isn't technically legal according to our definition of WFFs. What we'd like to show is that $\chi \wedge \upsilon \wedge \omega \vdash \chi \wedge \upsilon \wedge \omega$ and $\chi \wedge \upsilon \wedge \omega \vdash \chi \wedge \upsilon \wedge \omega$ as well as the equivalent for $\vee$. Thus, when we see three, four, or more terms in a conjunction (or disjunction), we can legitimately group them as we see fit.

Solution:

Here, we'll show only $\chi \wedge \upsilon \wedge \omega \vdash \chi \wedge \upsilon \wedge \omega$ and leave the other direction (and $\vee$'s associativity) to the reader. These are all very similar to the previous commutativity example.

1	$\chi \wedge \upsilon \wedge \omega$	Premise
2	$\chi \wedge \upsilon$	$\wedge$Elim (left), line 1
3	χ	$\wedge$Elim (left), line 2
4	υ	$\wedge$Elim (right), line 2
5	ω	$\wedge$Elim (right), line 1
6	$\upsilon \wedge \omega$	$\wedge$Intro, lines 4,5
7	$\chi \wedge \upsilon \wedge \omega$	$\wedge$Intro, lines 3,6

Note that we omitted the detailed explanation of how each rule applies, since this should be clear in each of these steps.

These deductions are straightforward and should be unsurprising, but perhaps not too interesting. These simple rules can carry us far and will be used commonly in other examples.

Example:
The case-elimination rule is easy enough for a dog! Rico has a vocabulary of over 200 words, and if asked to fetch an unknown toy, he can pick it out of a group of known toys by process-of-elimination. (It's almost enough to make you wonder whether dogs know calculus.)

This Border Collie knows his inference rules.

There is a subtle difference between implication ($\Rightarrow$) and provability ($\vdash$). Both embody the idea that the truth of the right-hand-side follows from the left-hand-side. But, $\Rightarrow$ is a syntactic formula connective combining two WFFs into a larger WFF, while $\vdash$ combines a list of propositions and a WFF into a statement about provability.

Exercise:

Problem:

Show that, $\varphi \vdash \psi$ is equivalent to $\vdash \varphi \Rightarrow \psi$ in that, we can show one if and only if we can show the other.

Solution:

First, if we know $\varphi \vdash \psi$, then that means there is some written proof… we know $\vdash \varphi \Rightarrow \psi$, simply by $\Rightarrow$Intro.

If we know $\vdash \varphi \Rightarrow \psi$, then if we add a premise φ, then ψ follows by $\Rightarrow$Elim.

Note how this proof is about other proofs! (However, while we reason about this particular inference system, we're not using this system while proving things about it — this proof is necessarily outside the inference system.

using subproofs
Using subproofs, to show RAA or to aid in presentation.

Subproofs

The reductio ad absurdum ("RAA"), Latin for "reduction to absurdity", seems very strange: If we can prove that false is true, then we can prove the negation of our premise. Huh!?! What on Earth does it mean to prove that false is true?

This is known as **proof-by-contradiction**. We start by making a single unproven assumption. We then try to prove that false is true. Clearly, that it nonsense, so we must have done something wrong. Assuming we didn't make any mistakes in the individual inference steps, then the only thing that could be wrong is the assumption. It must not hold. Therefore, we have just proven its negation.

This form of reasoning is often expressed via **contrapositive**. Consider the slogan " If you paid list price, you didn't buy it at SuperMegaMart. " (This is a contrapositive, because the real statement the advertisers want to make is that if you buy it at SuperMegaMart, then you won't pay list price.), which we'll abbreviate $\mathrm{payFull} \Rightarrow \neg\,\mathrm{boughtAtSMM}$. You know this slogan is true, and you just made a SuperMegaMart purchase ($\mathrm{boughtAtSMM}$), and are suddenly wanting a **proof** that you got a good deal. Well, suppose we didn't. That is, **suppose** $\mathrm{payFull}$. Then by the truth of the marketing slogan, we infer $\neg\,\mathrm{boughtAtSMM}$. But this contradicts $\mathrm{boughtAtSMM}$ (that is, from $\neg\,\mathrm{boughtAtSMM}$ and $\mathrm{boughtAtSMM}$ together we can prove that false is true). The problem must have been our pessimistic assumption $\mathrm{payFull}$; clearly that couldn't have been true, and we're happy to know that $\neg\,\mathrm{payFull}$.

Example:
Spot the proof-by-contradiction used in The Simpsons:
Bart, filing through the school records: " Hey, look at this: Skinner makes $25,000 per year! "
Other kids: "Ooooh!"
Milhouse: " And he's 40 years old; that makes him a millionaire! "
Skinner, indignantly: " I wasn't a principal when I was 1!"
Milhouse: " **And,** he paints houses during the summer … he's a billionaire! "
Skinner: " If I were a billionaire, would I still be living with my mother? " [Kids' laughter]
Skinner, to himself: " The kids just aren't responding to logic anymore! "

In the particular set of inference rules we have chosen to use, RAA is surprisingly important. It is the only way to prove formulas that begin with a single "¬". [footnote]
This is an example of reasoning **about** our logic system. It shows us that while we might have some redundant inference rules, RAA isn't one of them. The only other rule which produces formulas starting with an initial "¬" is ¬Intro. Is this also essential, or could we still prove all the same things even without ¬Intro?

Example:
We'll prove $\vdash \neg\,(\alpha \ \wedge \ \neg\alpha)$.

1	subproof:$\alpha \ \wedge \ \neg\alpha \vdash \mathrm{false}$		
1.a	$\alpha \ \wedge \ \neg\alpha$	Premise for subproof	

1.b		α	∧Elim (left), line 1.a, where $\varphi = \alpha$, and $\psi = \neg\alpha$	
1.c		$\neg\alpha$	∧Elim (right), line 1.a, where $\varphi = \alpha$, and $\psi = \neg\alpha$	
1.d		false	falseIntro, lines 1.b,1.c, where $\varphi = \alpha$	
2	$\neg(\alpha \wedge \neg\alpha)$			RAA, line 1, where $\varphi = \alpha \wedge \neg\alpha$

Exercise:

Problem:

Here's another relatively simple example which uses RAA. Show that the **modus tollens** rule holds: $\alpha \Rightarrow \beta, \neg\beta \vdash \neg\alpha$

Solution:

1	$\alpha \Rightarrow \beta$			Premise
2	$\neg\beta$			Premise
3	subproof:$\alpha \vdash$ false			
3.a		α	Premise for subproof	
3.b		β	⇒Elim, lines 1,3.a	
3.c		false	falseIntro, lines 2,3.b	
4	$\neg\alpha$			RAA, line 3

Another use of subproofs is to organize proofs' presentations. Many proofs naturally break down into larger subparts, each with its own intermediate conclusion. These steps between these subparts are big enough to correspond to our intuition, but too big to correspond to individual inference rules. This gives additional useful structure to a proof, aiding our understanding.

Example:
Previously, we showed that ∧ (AND) commutes. However, that conclusion is only directly applicable when the ∧ is at the "top-level", i.e., not nested inside some other connective. Here, we'll show that ∧ commutes inside ¬, or more formally, $\neg(\alpha \wedge \beta) \vdash \neg(\beta \wedge \alpha)$.

Note: When doing inference-style proofs, we will **not** use the Boolean algebra laws nor replace subformulas with equivalent formulas. Conversely, when doing algebraic proofs, don't use inference rules! While theoretically it's acceptable to mix the two methods, for homeworks we want to make sure you can do the problems using either method alone, so keep the two approaches separate!

We'll do two proofs of this to illustrate that there's always more than one way to prove something!
In our first proof, we'll use RAA. Why? Looking at our desired conclusion, what could be the last inference rule used in the proof to reach the conclusion? By the shape of the formula, the last step can't use any of the "introduction" inference rules (∧Intro, ∨Intro, ⇒Intro, falseIntro, or ¬Intro). We could potentially use any of the "elimination" inference rules. But, for ∧Elim, ∨Elim, ⇒Elim, ¬Elim, or CaseElim, we would first have to prove some more complicated formula to obtain our desired conclusion. That seems somewhat unlikely or unnecessary. For falseElim, we'd have to first prove false, i.e., obtain a contradiction, but our only premise isn't self-contradictory. The only remaining option is RAA.

1	$\neg(\alpha \wedge \beta)$		Premise
2	subproof:$\beta \wedge \alpha \vdash$ false		
2.a	$\beta \wedge \alpha$	Premise for subproof	
2.b	$\alpha \wedge \beta$	Theorem: ∧ commutes, line 2a	
2.c	false	falseIntro, lines 1,2.b	
3	$\neg(\alpha \wedge \beta)$		RAA, line 2

The proof above uses a subproof because it is necessary for the use of RAA. In contrast, the proof below uses two subproofs simply for organization.
For our second proof, let's not use RAA directly. Our plan is as follows:

- Assume the premise $\neg(\alpha \wedge \beta)$.
- Again, use commutativity to show that $\beta \wedge \alpha \Rightarrow \alpha \wedge \beta$
- Use modus tollens to obtain the conclusion.

We can organize the proof into corresponding subparts:

1	$\neg(\alpha \wedge \beta)$		Premise
2	subproof:$\beta \wedge \alpha \Rightarrow \alpha \wedge \beta$		
2.a	$\beta \wedge \alpha \vdash \alpha \wedge \beta$	Theorem statement: ∧ commutes	
2.b	$\beta \wedge \alpha \Rightarrow \alpha \wedge \beta$	⇒Intro, line 2.a	

3	subproof:$\neg(\beta \wedge \alpha)$		
3.a	$\beta \wedge \alpha \Rightarrow \alpha \wedge \beta, \neg(\alpha \wedge \beta) \vdash \neg(\beta \wedge \alpha)$	Theorem statement: modus tollens	
3.b	$(\beta \wedge \alpha \Rightarrow \alpha \wedge \beta) \wedge \neg(\alpha \wedge \beta) \Rightarrow \neg(\beta \wedge \alpha)$	⇒Intro, line 3.a	
3.c	$(\beta \wedge \alpha \Rightarrow \alpha \wedge \beta) \wedge \neg(\alpha \wedge \beta)$	∧Intro, lines 2,1	
3.d	$\neg(\beta \wedge \alpha)$	⇒Elim, lines 3.b,3.c	

More examples

Now let's use these rules in a couple larger proofs, to show some more interesting results.

Example:
Let's redo the first example's proof formally and show $\text{H-has-2} \wedge \text{J-safe} \vdash \text{G-unsafe}$. The inference rules we used informally above don't correspond exactly to those in our definition, so the formal proof is more complicated.

1	$\text{H-has-2} \Rightarrow \text{P-unsafe} \wedge \text{G-unsafe} \vee \text{J-unsafe} \wedge \text{P-unsafe} \vee \text{G-unsafe} \wedge \text{J-unsafe}$	
2	$\text{H-has-2} \wedge \text{J-safe}$	
3	H-has-2	
4	$\text{P-unsafe} \wedge \text{G-unsafe} \vee \text{J-unsafe} \wedge \text{P-unsafe} \vee \text{G-unsafe} \wedge \text{J-unsafe}$	
5	J-safe	
6	$\text{J-safe} \Rightarrow \neg\text{J-unsafe}$	
7	$\neg\text{J-unsafe}$	

8	subproof:G-unsafe $\wedge$ J-unsafe $\vdash$ false		
8.a		G-unsafe $\wedge$ J-unsafe	Pre for sub
8.b		J-unsafe	$\wedge$E (rig line
8.c		false	fals Intr line 7,8.
9	$\neg$ (G-unsafe $\wedge$ J-unsafe)		
10	P-unsafe $\wedge$ G-unsafe $\vee$ J-unsafe $\wedge$ P-unsafe		
11	subproof:J-unsafe $\wedge$ P-unsafe $\vdash$ false		
11.a		J-unsafe $\wedge$ P-unsafe	Pre for sub
11.b		J-unsafe	$\wedge$E (lef line
11.c		false	fals Intr line 7,1
12	$\neg$ (J-unsafe $\wedge$ P-unsafe)		
13	P-unsafe $\wedge$ G-unsafe		
14	G-unsafe		

Wow! This formalization is a lot longer than the original informal proof. That's a result of the particular set of inference rules we are using, that we can only make inferences in small steps. Also, here we were pickier about the distinction between "not safe" and "unsafe".

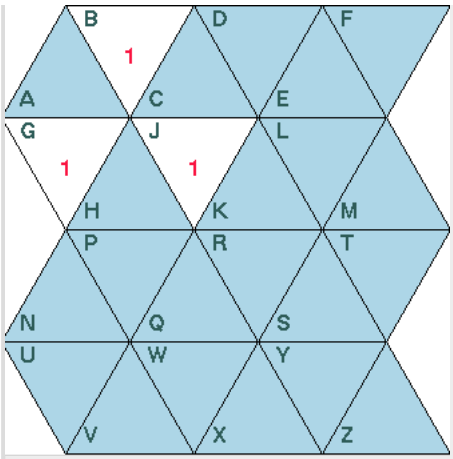
Example:
The previous example is a perfect candidate for adding structure to the proof by using additional subproofs. The following is more similar to the original informal proof.
Note also that subproofs can have their own subproofs.

1	H-has-2 ⇒ P-unsafe ∧ G-unsafe ∨ J-unsafe ∧ P-unsafe ∨ G-unsafe ∧ J-unsafe			
2	subproof:⊢ H-has-2			
2.a		H-has-2 ∧ J-safe		
2.b		H-has-2		
3	P-unsafe ∧ G-unsafe ∨ J-unsafe ∧ P-unsafe ∨ G-unsafe ∧ J-unsafe			
4	subproof:⊢ ¬J-unsafe			
4.a		H-has-2 ∧ J-safe		
4.b		J-safe		
4.c		J-safe ⇒ ¬J-unsafe		
4.d		¬J-unsafe		
5	subproof:⊢ P-unsafe ∧ G-unsafe			
5.a		subproof:G-unsafe ∧ J-unsafe ⊢ false		
5.a.i			G-unsafe ∧ J-unsafe	Pr fo su
5.a.ii			J-unsafe	∧ (ri lir 5.

5.a.iii			false	fa In lir 4,
5.b		$\neg$ (G-unsafe $\wedge$ J-unsafe)		
5.c		P-unsafe $\wedge$ G-unsafe $\vee$ J-unsafe $\wedge$ P-unsafe		
5.d		subproof:J-unsafe $\wedge$ P-unsafe $\vdash$ false		
5.d.i			J-unsafe $\wedge$ P-unsafe	Pr fo su
5.d.ii			J-unsafe	$\wedge$ (le lir 5.
5.d.iii			false	fa In lir 4,
5.e		$\neg$ (J-unsafe $\wedge$ P-unsafe)		
5.f		P-unsafe $\wedge$ G-unsafe		
6	G-unsafe			

A standard way of presenting proofs is by using **lemmas** to show parts of the proofs. Lemmas are simply formulas which we prove not as an end result, but as intermediate steps in a larger proof. So, they are simply another way of presenting subproofs.

Example:



Example WaterWorld board

Consider the above figure. We'll show B-has-1 ∧ G-has-1 ∧ J-has-1 ⊢ K-unsafe. We'll do this through the following series of lemmas:

- Lemma A: ¬A-unsafe,G-has-1 ⊢ H-unsafe
- Lemma B: ¬A-unsafe,B-has-1 ⊢ C-unsafe
- Lemma C: H-unsafe,C-unsafe,J-has-1 ⊢ false
- Lemma D: A-unsafe,B-has-1 ⊢ C-safe
- Lemma E: A-unsafe,G-has-1 ⊢ H-safe
- Lemma F: C-safe,H-safe,J-has-1 ⊢ K-unsafe

First, we'll show the main proof, assuming each of the lemmas. Then, proofs of each of the lemmas will follow.

1	B-has-1 ∧ G-has-1 ∧ J-has-1			Premise
2	B-has-1			∧Elim (left), line 1
3	G-has-1 ∧ J-has-1			∧Elim (right), line 1
4	G-has-1			∧Elim (left), line 3
5	J-has-1			∧Elim (right), line 3
6	subproof:¬A-unsafe ⊢ false			
6.a		¬A-unsafe	Premise for subproof	
6.b		H-unsafe	Lemma A, lines 6.a,4	
6.c		C-unsafe	Lemma B, lines 6.a,2	
6.d		false	Lemma C, lines 6.b,6.c,5	

7	A-unsafe		RAA, line 6
8	C-safe		Lemma D, lines 7,2
9	H-safe		Lemma E, lines 7,3
10	K-unsafe		Lemma F, lines 8,9,5

And that's the desired proof! Now it just remains to show each of the six lemmas.
Lemma A: ¬A-unsafe,G-has-1 ⊢ H-unsafe

1	¬A-unsafe			Premise
2	G-has-1			Premise
3	subproof:A-unsafe ∧ H-safe ⊢ false			
3.a		A-unsafe ∧ H-safe	Premise for subproof	
3.b		A-unsafe	∧Elim	
3.c		false	falseIntro, lines 1,3b	
4	¬ (A-unsafe ∧ H-safe)			RAA, line 3
5	G-has-1 ⇒ A-safe ∧ H-unsafe ∨ A-unsafe ∧ H-safe			WaterWorld axiom
6	A-safe ∧ H-unsafe ∨ A-unsafe ∧ H-safe			⇒Elim, lines 5,2
7	A-unsafe ∧ H-safe ∨ A-safe ∧ H-unsafe			Theorem: ∨ commutes, line 6
8	A-safe ∧ H-unsafe			CaseElim, lines 4,7
9	H-unsafe			∧Elim (right), line 8

Lemma B: ¬A-unsafe,B-has-1 ⊢ C-unsafe

1	$\neg$A-unsafe			Premise
2	B-has-1			Premise
3	subproof:A-unsafe $\wedge$ C-safe $\vdash$ false			
3.a		A-unsafe $\wedge$ C-safe	Premise for subproof	
3.b		A-unsafe	$\wedge$Elim (left), line 3a	
3.c		false	falseIntro, lines 1,3b	
4	$\neg$ (A-unsafe $\wedge$ C-safe)			RAA, line 3
5	B-has-1 $\Rightarrow$ A-safe $\wedge$ C-unsafe $\vee$ A-unsafe $\wedge$ C-safe			WaterWorld axiom
6	A-safe $\wedge$ C-unsafe $\vee$ A-unsafe $\wedge$ C-safe			$\Rightarrow$Elim, lines 5,2
7	A-unsafe $\wedge$ C-safe $\vee$ A-safe $\wedge$ C-unsafe			Theorem: $\vee$ commutes, line 6
8	A-safe $\wedge$ C-unsafe			CaseElim, lines 4,7
9	C-unsafe			$\wedge$Elim (right), line 8

Proving the other lemmas is left as an exercise to the reader.

Note that we took a little shortcut: we used the lemmas as if they were inference rules. According to our previous definition of proofs, we technically should present the lemma as a subproof and then use an inference rule or two to show how that applies, as we've done in previous examples. This shorter form is common practice and much easier to read.

In summary, we must state one of the following four possible reasons for each step in a proof, allowing subproofs.

- This step's WFF is a premise.
- This step's WFF is an axiom.
- This step's WFF follows from a inference rule applied to previous steps' WFFs. The reason includes a statement of which inference rule is used and how.
- This step's WFF follows from a subproof, where that subproof may temporarily introduces additional premises. The reason includes the entire subproof. When that subproof has been shown elsewhere, such as in class or another exercise, it may simply be cited, for brevity. Of course, subproofs may have additional embedded subproofs, in turn.

Technically, when using subproofs, one must be careful to rename variables, to avoid clashes. Rather than formalize this notion, we'll leave it as "obvious".

Propositional Logic: soundness and completeness revisited

"The folly of mistaking a paradox for a discovery, a metaphor for a proof, a torrent of verbiage for a spring of capital truths, and oneself for an oracle, is inborn in us. — Paul Valery, poet and philosopher (1871-1945)"

Throughout this discussion, we've implicitly assumed that if we've proven something, it must be true. But we should be careful: What if one of those listed inference rule isn't always valid? What if we introduced a new rule? (Sure, you'd probably balk if we proposed something silly like $a \lor b \Rightarrow a$, or even more degenerately false. But what about some more reasonable-sounding rule?) What if our new rule introduces an inconsistency, when combined with the other rules in a some complicated way? In fact, are we **absolutely certain** that this can't already happen with the inference rules we have?! This brings us back to the questions of soundness and completeness of a proof system. Fortunately, the system presented here is both sound and complete (though proving this is beyond our current scope). However, we can rest assured, that for propositional logic, what we can **prove** really does correspond entirely to what is **true**.

Exercise:

> **Problem:**
>
> If we omitted the RAA inference rule, would this new system be sound? Would it be complete?

> **Solution:**
>
> It would be sound: Look at all the possible proofs that can be made in the original system; all those proofs lead to true conclusions (since that original system is sound, as we're claiming). If we just discard all those that include RAA, the remaining proofs are still all true, so the smaller system is sound.
>
> It would not be complete, though: As pointed out, RAA is our only way to prove negations without premises. There are negated formulas that are true (and have no premises) — for example $\neg\,\mathrm{false}$. Without

RAA, we cannot provide a proof of $\neg$ false, so the smaller system is **incomplete**.

Proofs and programming

Proofs are organized a lot like programs. Based on some premises (inputs), we obtain some conclusion (output) after using a series of inference rules (basic computation like addition and other operations). Using subproofs, especially when citing previous proofs, is just like organizing our program into functions that can be used many times.

Naturally, since using inference rules is not only how people prove things, but also computers. A clear example is in type checking. The core idea of type checking a function application is "If function `f` takes an argument of type α and producing an output of type β, and expression `exp` is of type α, then `f(exp)` is of type β." This type rule closely resembles $\Rightarrow$Elim: "If a proven formula is $a \Rightarrow b$ and other proven formula is a, then together, b is a proven formula." Furthermore, this similarity is highlighted by notation in many programming languages which would write the type of `f` as $\alpha \rightarrow \beta$. Type rules are simply inference rules for proving results about the types of programs, and in most typical programming languages these rules closely correspond to those we are using for logic. This correspondence is known as the Curry-Howard Isomorphism.

As with logic, we want type checkers to be sound and complete. Soundness here means that if the program passes type checking, when we execute the program (or single function) and get a value, that value is of the stated type. In other words, if our program type checks, then we are guaranteed that some kinds of errors will not happen at run-time. That also means that if our program would have a run-time type error, the type checker will correctly report that our program is erroneous. Completeness here means that if we execute the program (or single function) and get a value of a certain type, then our type checker indeed tells us that type.

Note that type checking is still an area of active research, since the job is made difficult in the presence of language features such as inheritance, multiple inheritance, dynamic class loading, etc. When people introduce new computer languages with new features, and want to claim that their

new language is **type safe** (that no function ever will be applied to the wrong type at run-time), then the paper which introduces the language will contain such a proof.

Propositional Logic: conclusions

Are we done yet?

These inference rules may seem limited, and you may have some more general ones in mind. Soon, we'll see additional inference rules in the context of first-order logic, which will give us a richer set of proofs. In general, a hard problem is finding a language that is both expressive enough to describe the domain succinctly, but also limited enough to automate reasoning. This is a very practical issue in type checking and other program analysis. While it can be easy to find some program errors automatically, it is very difficult or impossible to **guarantee** that you can find **all** errors (of some specific kind, like type errors).

One thing we would like to eliminate is the need (at least technically) to restate structurally identical proofs, as discussed for commutativity. We will be able to add the idea of generalizing such proofs directly into the logic and inference rules.

Despite the desire for more flexible reasoning, we'd also like to consider whether we have more inference rules than are necessary. Are some of them redundant? This is similar to the software rule that we should have a single point of control, or the similar idea that libraries should provide exactly one way of doing something. In general, this is not easy to ensure. We have shown that some potential additional inference rules, like commutativity and associativity, weren't necessary. But we haven't shown our core inference rules to be minimal. What do you think? (See the homework exercise problems on the redundancy of not-elimination, not-introduction, and case-elimination.)

Distinctness of the approaches (optional)

You might be wondering — can we use propositional equivalences as axioms when using inference rules? The short answer is no. First, Boolean equivalences are **pairs** of formulas, whereas axioms are individual formulas. Second, none of our inference rules mention equivalences.

However, let's reword the question — could we use propositional equivalences when using inference rules? It would make sense to add an inference rule to allow this. One possibility would be an inference rule that turns an equivalence into an implication: “if we know , then we know .” Another possibility would be an inference rule that allows us to substitute equivalence subterms, as we do in equivalence proofs: “if we know and , then we know , i.e., , except with instances of replaced by .” With either, we would also have to allow equivalence proofs as subproofs or lemmas in inference proofs.

Traditionally, and in our presentation, we **do not** combine equivalences and inference rules in any such way. The disadvantage of combining them is that instead of two relatively simple proof systems, you would have one more complicated proof system. It would be harder to learn all that you could do in such a system, and for theorists, it would be harder to prove things such as soundness and completeness for the combined system. In learning and describing proofs, it is best to keep them separate. However, the advantage would be shorter proofs. When **using** the combined system, you'd have flexibility to use whichever technique suited the current step best. In practice, people commonly combine these and other proof techniques.

Exercises for Propositional Logic I
Problems on propositional logic, including truth-tables, boolean algebra, and inference rules.

Please write logic formulas using the syntax previously defined, using false (or for brevity, "F"), true (or "T"), ¬, ∧, ∨, and ⇒. Except where directed, use only these connectives.

Note: You can download WaterWorld if you like. At Rice University, WaterWorld is installed on OwlNet, in `/home/comp280/bin/waterworld`.

Propositional Logic

Exercise:

> **Problem:**[Practice problem—solution provided.]
>
> Your friend Tracy argues: " It is bad to be depressed. Watching the news makes me feel depressed. Thus, it's good to avoid watching the news. "
>
> Regardless of whether the premises and conclusion are true, show that the **argument** is not, by showing it doesn't hold for all domains. Replace "depressed" and "watching news" with expressions which leave the premises true, but the conclusion false (or at least, what most reasonable people would consider false).

> **Solution:**
>
> Lots of possible counterexamples. " It is bad to be depressed. Doing homework makes me depressed; so it's good to not do my homework. " Or, " It is bad for people to be in physical pain. Childbirth causes pain. Therefore childbirth needs be avoided by all people. " If the

original conclusion is really correct, Tracy needs to elucidate some of his unspoken assumptions.

The flaw seems to be along the lines of, " avoiding bad in the short run may not always be good in the long run " (or equivalently, sometimes you have to choose the lesser of two evils). No, you weren't asked to name a specific flaw, and reasonable people can differ on precisely what the flaw is. (And, formal logic is not particularly helpful here.) Nonetheless, uncovering hidden assumptions in arguments often helps understand the real issues involved.

Note: For fun, pick up the front page of the daily newspaper, and see how many arguments use faulty rules of inference and/or rely on unspoken premises (which not all might agree with). In particular, political issues as spun to the mainstream press are often riddled with error, even though there are usually reasonable arguments on both sides which policy-makers and courts debate.

Exercise:

Problem:[Practice problem—solution provided.]

An acquaintance says the following to you: " Chris claims knowledge is more important than grades. But she spent yesterday doing an extra-credit assignment which she already knew how to do. Therefore, she's a hypocrite and deserves no respect. "

Regardless of whether the premises and conclusion are true, show that the **argument** is not, by showing it doesn't hold for all domains. Replace "knowledge" and "grades" with expressions which give you true premises, but a false conclusion (or at least, what most reasonable people would consider false).

Note: Exaggerate "knowledge" to something more important, and "grades" to something less important.

Solution:

" Terry claims that encouraging human-rights is more important than playing Tetris. But Terry played Tetris yesterday rather than volunteering with Amnesty International. " Most people wouldn't condemn Terry as a hypocrite just because of this; even the most dedicated of people are entitled to **some** free time. If your friend wants to prove Terry hypocritical, they'll have to provide further evidence or arguments.

Or similarly, " Politician X **claims** to support science funding, but voted against a proposal to shift all Medicare funds to NASA. "

Exercise:

Problem:[Practice problem—solution provided.]

While the following argument may sound plausible initially, give a particular situation where the conclusion doesn't hold (even though the premises do). Then, in a sentence or two, sketch why your counterexample may still represent rational behavior by pointing out a real-world subtlety that the initial argument ignored.

1. If a certain outfit meets a dress code, then per force all less-revealing outfits also meet that dress code.

2. In public transportation projects, out of two alternatives, the cheaper one which gets the job done is the better choice.

Solution:

1. It can be socially acceptable to wear my swimsuit into a fast-food restaurant. My underwear is less revealing than my swimsuit, and yet it would still raise many more eyebrows to go to that restaurant in my underwear, than my swimsuit.

   Clothes (and style in general) somehow encompass a form of communication, and people may object to an outfit's mood or message without actually objecting to how much the outfit reveals. (Other examples of communication-through-style include team logos, t-shirts with humorous slogans, and arm bands.)

2. Buses are a lot cheaper than light rail. Yet, the light-rail here in Houston demonstrates that many people who wouldn't routinely take a bus **are** willing to take light rail. (Only after we recognize this, can we try to figure out what **why** the difference exists, and then brainstorm to find a better overall solution.)

Exercise:

Problem:

Choose **just one** of the following informal arguments. While the argument sounds plausible initially, give a particular situation where the conclusion doesn't hold (even though the premises do). Then, briefly state why your counterexample may still represent rational behavior by pointing out a real-world subtlety that the initial argument ignored.

1. [cell phone] Talking on a cell phone while driving increases the likelihood of an accident. Interestingly, hands-free phones do not significantly help. It's just the distraction of a phone conversation that causes the problem.

2. [equivalent products] If two companies offer two materially equivalent products, then most everybody will buy the cheaper one.

3. [service] In a free market, if a company doesn't offer good service, individual customers will become fed up and take their business elsewhere.

4. [web browser] If there are two versions of a free web browser, and they run equally quickly, users will use the one with better features/interface.

5. [door-locking] Anybody who really wants to break into your house while you're gone will be able to. (For instance, using a towel to muffle sound, break the corner of a back window, reach in and unlatch the window, and climb through.) So there's no point in locking your front door.

Exercise:

Problem:[Practice problem—solution provided.]

Let p, q, and r be the following propositions:

- p: You get an A on the final exam
- q: You do every exercise in the book.
- r: You get an A in this class.

Write the following formulas using p, q, and r and logical connectives.

1. You get an A in this class, but you do not do every exercise in the book.

2. To get an A in this class, it is necessary for you to get an A on the final.

3. Getting an A on the final and doing every exercise in the book is sufficient for getting an A in this class.

Solution:

1. $r \ \wedge \ \neg q$
2. $r \Rightarrow p$

   Think of the English being reworded to " If you got an A in this class, you must have gotten an A on the final. "

3. $p \ \wedge \ q \Rightarrow r$

Exercise:

Problem:

Translate the following English sentences into propositional logic. Your answers should be WFFs.

1. If the Astros win the series ("AW"), then pigs will fly ("PF").
2. Pigs will not fly, and/or bacon will be free ("BF").
3. The Astros will win the series, or bacon will be free, but not both.

Exercise:

Problem:[Practice problem—solution provided.]

It just so happens that all the web pages in Logiconia which contain the word "Poppins" also contain the word "Mary". Write a formula (a query) expressing this. Use the proposition $\mathrm{Poppins}$ to represent the concept "the web page contains 'Poppins'" (and similar for Mary).

Solution:

$\mathrm{Poppins} \Rightarrow \mathrm{Mary}$

Exercise:

Problem:

- If a Logicanian page contains the word "weasel", then it also contains either "words" or "eyed"; and
- Whenever a Logiconian page contains the word "mongoose", it does **not** also contain the word "weasel"; and
- Finally, all Logiconian pages contain the word "Logiconia", rather patriotically.

Write a formula expressing all this. (Your formula will involve five propositions: weasel, words, … Try to find a formula which mirrors the wording of the English above.)

Given the above statements, if a web page in Logiconia does not contain "weasel", does it contain "mongoose"?

Let's go meta for a moment: Is **this** web page Logiconian? (Yes, this one you're looking at now, the one with the homework problems.) Explain why or why not.

Exercise:

Problem:

Different search engines on the web have their own syntax for specifying searches.

Note: Note that a formula may be true for some web pages, and false for others. The search engine is concerned with finding all web pages which satisfy the formula. This is called a **query**, in database lingo.

Only a few allow full Boolean queries. Some interpret a list of several words in a row as an implicit conjunction, others as an implicit

disjunctions.

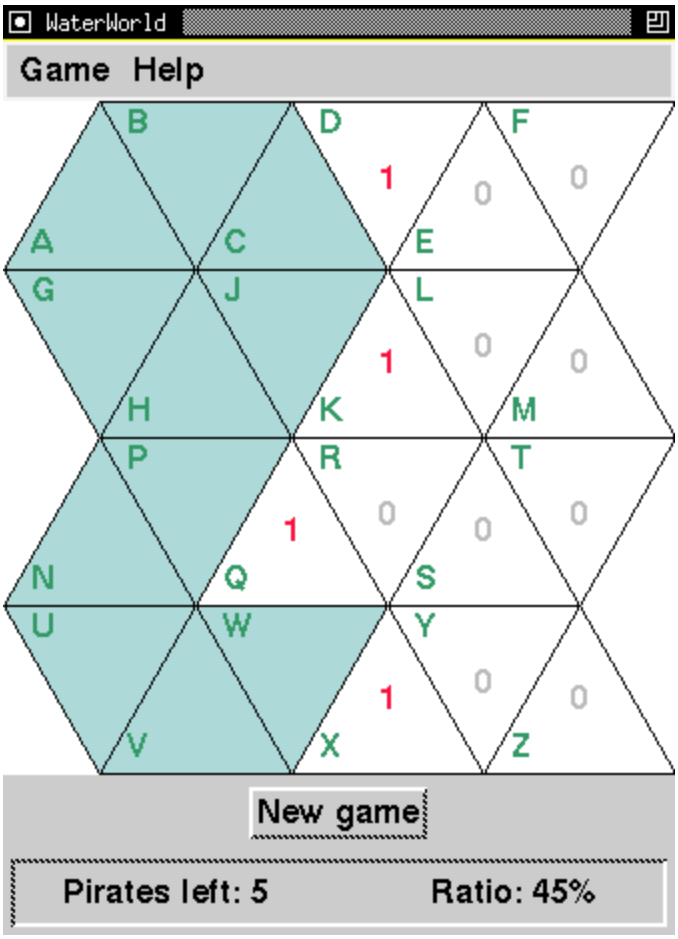
1. Read about the search syntax for the search language of eBay®. Write an eBay query for auctions which contain "border", do not contain "common", and contain at least one of "foreign" or "foriegn" [sic, misspellings are a great way to find underexposed auctions].

2. Google£'s advanced search is typical for the online search engines. In particular, you can search for results containing **all of** a, b, …, **at least one of** c, d, …, and **none of** e, f, …. Describe how that corresponds to a Boolean formula.

3. Give an example of a Boolean formula which cannot be rewritten to conform to Google's advanced search interface.

Exercise:

Problem:[Practice problem—solution provided.]



A sample WaterWorld board

Consider the particular board shown in the above figure.

1. Y-safe, Y-has-0, and ¬Y-has-2 are among the formulas which are true for this board but not for all boards. That is, they are neither domain axioms nor tautologies. Give two other such formulas.

2. V-safe might or might not be true for this board. Give two other such formulas.

Solution:

1. There are many simple answers, such as Y-has-1, $\neg\text{W-has-1}$, …

2. There are many simple answers, such as a, N-has-1, J-has-3, …

For each, there are also many such formulas composed with connectives such as $\wedge$ and $\vee$.

Exercise:

Problem:

In that same board, is location W safe? What is your informal reasoning? (List all your small steps.) Similarly for location P.

Exercise:

Problem:

Give a domain axiom of WaterWorld which was omitted in the ellipses in the WaterWorld domain axioms.

Exercise:

Problem:

Even allowing for ellision, the list of WaterWorld domain axioms is incomplete, in a sense. The game reports how many pirates exist in total, but that global information is not reflected in the propositions or axioms.

First, assume we only use the default WaterWorld board size and number of pirates, i.e., five. Give samples of the additional axioms that we need.

Next, generalize your answer to model the program's ability to play the game with a different number of pirates.

Exercise:

Problem: Give one WFF which meets all three conditions:

- true in all WaterWorld boards ("A **theorem** of WaterWorld")
- not already listed as one of the WaterWorld domain axioms, and
- not a tautology of propositional logic (can be made false in some truth assignment, though it may not be a truth assignment which satisfies the waterworld axioms).

Reasoning with Truth Tables

When writing truth tables, please list rows in the order used in all examples: FF, FT, TF, TT. For three-input tables, use the above four lines preceded by F, then the above four lines preceded by T.

Exercise:

Problem:

In a truth table for two inputs, provide a column for each of the sixteen possible distinct functions. Give a **small** formula for each of these functions.

> **Note:** These functions will include those for ∧, ∨, and the other connectives whose truth tables you've already seen.

Exercise:

Problem:[Practice problem—solution provided.]

Write the truth table for **xnor**, the negation of exclusive-or, What is a more common name for this Boolean function?

Solution:

φ	ψ	φ xnor ψ
false	false	true
false	true	false
true	false	false
true	true	true

Truth table for xnor

This is the "equals" for Booleans. It is also represented by the connective if-and-only-if.

If you said something like "the both-or-neither function", that's not quite good enough, as it's a roundabout way of expressing the simple idea of equivalence. Granted, it takes some practice to internalize Booleans as values, and that equality is as valid for them as for any other value.

Exercise:

Problem:

How many years would it take to build a truth table for a formula with 1000 propositions? Assume it takes 1 nanosecond to evaluate each formula.

A formula with 1000 propositions clearly isn't something you would create by hand. However, such formulas easily arise when modeling

the behavior of a program with a 1000-element data structure.

Exercise:

Problem:

Use truth tables to answer each of the following. Showing whether the connectives obey such properties via truth tables is one way of establishing which equivalences or inference rules we should use.

1. Show whether ⇒ is commutative.
2. Show whether ⊕ is commutative.
3. Show whether ⊕ is associative.

Prove that ∧ distributes over ∨:

4. $\varphi \ \wedge \ (\psi \ \vee \ \theta) \equiv \varphi \ \wedge \ \psi \ \vee \ \varphi \ \wedge \ \theta$

> **Note:** This version is left-distributivity. Right-distributivity follows from this plus the commutativity of ∧.

Prove that ∨ distributes over ∧:

5. $\varphi \ \vee \ \psi \ \wedge \ \theta \equiv (\varphi \ \vee \ \psi) \ \wedge \ (\varphi \ \vee \ \theta)$
6. Show whether ∧ or ∨ distribute over ⇒.
7. Show whether ⇒ distributes over ∧ or ∨.
8. Show whether ∧ or ∨ distribute over ⊕.
9. Show whether ⊕ distributes over ∧ or ∨.

Exercise:

Problem:

For each of the following, find a satisfying truth assignment, (values of the propositions which make the formula true), if any exists.

1. $(a \Rightarrow \neg b) \ \wedge \ a$
2. $(a \Rightarrow c \Rightarrow \neg b) \ \wedge \ (a \ \vee \ b)$

Exercise:

Problem:

For each of the following, find a falsifying truth assignment, (values of the propositions which make the formula false), if any exists.

1. $(a \Rightarrow \neg b) \ \vee \ a$
2. $(\neg b \Rightarrow (a \Rightarrow c)) \ \vee \ a \ \wedge \ b$

Exercise:

Problem:

Formula φ is **stronger** than formula ψ if ψ is true whenever φ is true (i.e., φ is **at least a strong** as ψ), but not conversely. Equivalently, this means that $\varphi \Rightarrow \psi$ is always true, but $\psi \Rightarrow \varphi$ is not always true.

As one important use of this concept, if we know that $\psi \Rightarrow \theta$, and that φ is stronger than ψ, then we also know that $\varphi \Rightarrow \theta$. That holds simply by transitivity. Another important use, which is outside the scope of this module, is the idea of strengthening an inductive hypothesis.

Similarly, φ is **weaker** than formula ψ whenever ψ is stronger than φ.

Show which of the following hold. When true, show $\varphi \Rightarrow \psi$ is true by a truth table, and show a falsifying truth assignment for $\psi \Rightarrow \varphi$. When false, give a truth table and truth assignment the other way around.

1. $a \ \wedge \ b$ is stronger than $a \ \vee \ b$.

2. $a \ \vee \ b$ is stronger than a.

3. a is stronger than $a \Rightarrow b$.

4. b is stronger than $a \Rightarrow b$.

Exercise:

Problem:

Using truth tables, show that $(a \ \vee \ c) \ \wedge \ (b \Rightarrow c) \ \wedge \ (c \Rightarrow a)$ is equivalent to $(b \Rightarrow c) \ \wedge \ a$. but not equivalent to $(a \ \vee \ c) \ \wedge \ (b \Rightarrow c)$.

Exercise:

Problem:[Practice problem—solution provided.]

When writing a complicated conditional that involves multiple pieces of data, it is easy to incorrectly oversimplify. One strategy for avoid mistakes is to write such code in a two-step process. First, write a conditional with a case for **every** possible combination, as in a truth table. Second, simplify the conditional.

Using this approach, we might obtain the following code after the first step. Simplify this code.

```
list merge_sorted_lists(list list1, list list2)
{
   if (is_empty(list1) && is_empty(list2))
      return empty_list;
   else if (is_empty(list1) &&
!is_empty(list2))
      return list2;
```

```
   else if (!is_empty(list1) &&
is_empty(list2))
      return list1;
   else if (!is_empty(list1) &&
!is_empty(list2)) {
      if (first_element(list1) <
first_element(list2))
         return make_list(first_element(list1),

merge_sorted_lists(rest_elements(list1),list2))
;
      else if (first_element(list1) >=
first_element(list2))
         return make_list(first_element(list2),

merge_sorted_lists(list1,rest_elements(list2)))
;
   }
}
```

Solution:

```
list merge_sorted_lists(list list1, list list2)
{
   if (is_empty(list1))
      return list2;
   else if (is_empty(list2))
      return list1;
   else {
      if (first_element(list1) <
first_element(list2))
         return make_list(first_element(list1),

merge_sorted_lists(rest_elements(list1),list2))
```

```
;
      else
         return make_list(first_element(list2),

merge_sorted_lists(list1,rest_elements(list2)))
;
   }
}
```

Alternatively, we could test the emptiness of the lists in the other order.

Exercise:

Problem:

Consider the following conditional code, which returns a boolean value.

```
int  i;
bool a,b;

…

if (a && (i > 0))
   return b;
else if (a && i <= 0)
   return false;
else if (a || b)
   return a;
else
   return (i > 0);
```

Simplify it by filling in the following blank with a single Boolean expression. Do not use a conditional (such as `if` or `?:`).

```
int  i;
bool a,b;

…

return ____________________;
```

Use either Java/C++ or Scheme syntax. In the former case, please fully parenthesize to make your formula unambiguous, rather than relying on Java's or C++'s many levels of operator precedence.

Reasoning with Equivalences

Exercise:

Problem:[Practice problem—solution provided.]

Using algebraic identities, and the definition or **nor** (mnemonic: "not or"), written $\downarrow$, $\varphi \downarrow \psi \equiv \neg (\varphi \vee \psi)$, express the function $\wedge$ in terms of $\downarrow$ only. That is, give a formula which doesn't use $\wedge$, $\vee$, $\neg$, but instead only uses $\downarrow$ and which has the same truth table as $\varphi \wedge \psi$.

Solution:

First we show that we can write negation in terms of $\downarrow$, or more specifically, $\neg\theta \equiv \theta \downarrow \theta$. Checking this on a truth table is pretty easy (there are only two rows to check). But for this question we need to use algebraic manipulation. This can be derived in a couple of simple steps:

1	$\neg\theta$	
2	$\equiv \neg\theta \ \wedge \ \neg\theta$	Idempotency of $\wedge$
3	$\equiv \neg\left(\theta \ \vee \ \theta\right)$	DeMorgan's law
4	$\equiv \theta \downarrow \theta$	Definition of nor

We use this lemma to show our ultimate goal:

1	$\delta \ \wedge \ \varepsilon$	
2	$\equiv \neg\neg\left(\delta \ \wedge \ \varepsilon\right)$	Double Complementation
3	$\equiv \neg\left(\neg\delta \ \vee \ \neg\varepsilon\right)$	DeMorgan's law
4	$\equiv \neg\left(\left(\delta \downarrow \delta\right) \ \vee \ \neg\varepsilon\right)$	Lemma, with $[\theta \mapsto \delta]$
5	$\equiv \neg\left(\left(\delta \downarrow \delta\right) \ \vee \ \left(\varepsilon \downarrow \varepsilon\right)\right)$	Lemma, with $\theta = \varepsilon$
6	$\equiv \delta \downarrow \delta \downarrow \varepsilon \downarrow \varepsilon$	Definition of nor, where $\varphi = \delta \downarrow \delta$, and $\psi = \varepsilon \downarrow \varepsilon$

Note that we judiciously used new meta-variables δ and ε rather than re-using φ and ψ (which would still be correct, but would make the graders need to pay much closer attention to the scope of those variables).

Exercise:

Problem:

Similar to the previous exercise, express each of the following using nand only, and prove correctness using the algebraic identities.

This operation is particularly interesting, since making a NAND gate in hardware requires only two transistors.

1. ¬
2. ∧
3. ∨

Exercise:

Problem:

Using algebraic identities, show that
$(a \lor c) \land (b \Rightarrow c) \land (c \Rightarrow a)$ is equivalent to $(b \Rightarrow c) \land a$.

This is an algebraic hand-evaluation: a series of formulas joined by $\equiv$. Don't write just portions of previous formulas and mysteriously re-introduce the dropped parts later. For each step, mention which identity you used. It is also helpful if you underline the formula you are rewriting in the next step. You can use commutativity and associativity without using a separate line, but mention when you use it.

Exercise:

Problem:

In two exercises, you've shown the same equivalence by truth tables and by algebraic identities.

1. What is an advantage of using truth tables? What is an advantage of using identities?

2. In that truth table exercise, you also showed two formulas φ and ψ **non**-equivalent. It is also possible to do so with Boolean algebra rather than truth tables. How?

3. Describe a hybrid approach, combining truth tables and Boolean algebra, to prove the equivalence and non-equivalence of formulas.

4. To ponder on your own without turning it in: Which approach appeals more to you?

Exercise:

Problem:

Using algebraic identities, rewrite the formula $(a \Rightarrow b \ \vee \ c) \ \wedge \ \neg b$ to one with fewer connectives.

Exercises for Propositional Logic II

Reasoning with Inference Rules

For proofs on this homework, remember that each step must be justified by one of the following:

- a premise,
- a WaterWorld axioms,
- a listed inference rule with the referenced line numbers (and, if ambiguous, substitutions for the inference rule's meta-variables), or
- a subproof shown inline, or equivalently, a theorem/lemma shown previously.

Except where otherwise directed, you may use any theorem shown in the text or by a previous exercise, even if that exercise was not assigned.

Exercise:

Problem: Fill in the blank reasons in the following proof that ∨ commutes, that is, $\chi \vee \upsilon \vdash \upsilon \vee \chi$.

1	$\chi \vee \upsilon$		Premise
2	subproof:$\chi \vdash \upsilon \vee \chi$		
2.a	χ	Premise for subproof	
2.b	$\upsilon \vee \chi$	∨Intro, line 2.a	
3	subproof:$\upsilon \vdash \upsilon \vee \chi$		
3.a	υ	Premise for subproof	
3.b	$\upsilon \vee \chi$	_____	
4	$\upsilon \vee \chi$		_____

Exercise:

Problem: Show that $\varphi \wedge \psi, \varphi \Rightarrow \theta, \psi \Rightarrow \delta \vdash \theta \wedge \delta$.

Note: It should take around 8 steps.

Exercise:

Problem: Show what is often called the implication chain rule: $\varphi \Rightarrow \psi, \psi \Rightarrow \theta \vdash \varphi \Rightarrow \theta$.

Exercise:

Problem:[Practice problem—solution provided.]

Show what is often called negated-or-elimination (left): $\neg(\varphi \vee \psi) \vdash \neg\varphi$.

Note: Think backwards. How can we end with $\neg\varphi$? One way is to end with RAA, under the premise φ. Using that premise φ and the starting premise $\neg(\varphi \vee \psi)$ can you derive the contradiction?

Solution:

1	$\neg(\varphi \vee \psi)$			Premise
2	subproof:$\varphi \vdash$ false			
2.a		φ	Premise for subproof	
2.b		$\varphi \vee \psi$	∨Intro, line 2a	
2.c		false	falseIntro, lines 1,2b	
3	$\neg\varphi$			RAA, line 2

Exercise:

Problem: Using the inference rule RAA, prove $\neg\varphi \vdash \neg(\varphi \wedge \psi)$.

Exercise:

Problem: Show that $\neg$W-safe $\vee$ $\neg$Y-unsafe $\vdash$ W-unsafe $\vee$ Y-safe.

Note: The proof is a bit longer than you might expect. Use the ∨Elim inference rule to get the final result.

Exercise:

Problem:

In our inference rules, unlike our equivalences, we chose to not include any corresponding to distributivity.

1. Prove a left-hand version of one direction of distributivity: $\varphi \wedge (\psi \vee \theta) \vdash \varphi \wedge \psi \vee \varphi \wedge \theta$.

Use the previous part's result, plus ∧'s commutativity to prove the corresponding right-hand version:

2. $(\psi \vee \theta) \wedge \varphi \vdash \psi \wedge \varphi \vee \theta \wedge \varphi$.

Exercise:

Problem:

In our inference rules, unlike our equivalences, we chose to not include any corresponding to DeMorgan's Law. Show that each of the following versions is still provable.

1. $\varphi \vee \psi \vdash \neg(\neg\varphi \wedge \neg\psi)$
2. $\neg(\varphi \vee \psi) \vdash \neg\varphi \wedge \neg\psi$
3. $\varphi \wedge \psi \vdash \neg(\neg\varphi \vee \neg\psi)$
4. $\neg(\varphi \wedge \psi) \vdash \neg\varphi \vee \neg\psi$

Exercise:

Problem:

The above exercise suggests that it would be useful to have an inference rule or theorem that says given $\theta \vdash \neg\delta$, then $\neg\theta \vdash \delta$. Or, equivalently, because of ⇒Intro and ⇒Elim, $\theta \Rightarrow \neg\delta \vdash \neg\theta \Rightarrow \delta$. Why don't we?

Exercise:

Problem:

In our inference rules, unlike our equivalences, we have nothing that directly equates $\varphi \Rightarrow \psi$ and $\neg\varphi \vee \psi$. Prove each of the following.

1. $\varphi \Rightarrow \psi \vdash \neg\varphi \vee \psi$
2. $\neg\varphi \vee \psi \vdash \varphi \Rightarrow \psi$

Exercise:

Problem: Prove the following: $\varphi \Rightarrow \psi, \psi \Rightarrow \varphi \vdash \varphi \wedge \psi \vee \neg\varphi \wedge \neg\psi$

Exercise:

Problem: Prove what is commonly called the **Law of Excluded Middle**: $\vdash \chi \vee \neg\chi$.

1. Give a short proof citing our previous proof of $\vdash \neg(\chi \wedge \neg\chi)$ and the relevant version of DeMorgan's Law from above.

2. Give a direct version without using previous theorems.

> **Note:** Use RAA two or three times.

Exercise:

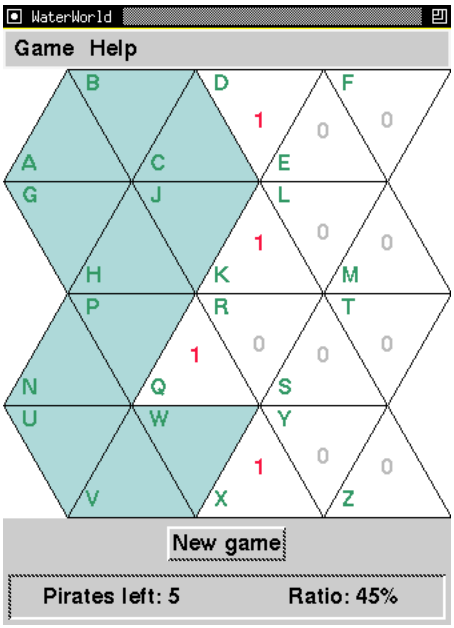
Problem:

Prove the missing steps and reasons in the following WaterWorld proof of
X-has-1 ⊢ W-unsafe ∨ Y-unsafe.

1	X-has-1		
2	_____		
3	_____		
4	subproof: $\text{W-safe} \ \wedge \ \text{Y-unsafe} \vdash \text{W-unsafe} \ \vee \ \text{Y-unsafe}$		
4.a		$\text{W-safe} \ \wedge \ \text{Y-unsafe}$	Premise for subproof
4.b		Y-unsafe	_____
4.c		$\text{W-unsafe} \ \vee \ \text{Y-unsafe}$	_____
5	subproof: $\neg \left(\text{W-safe} \ \wedge \ \text{Y-unsafe}\right) \vdash \text{W-unsafe} \ \vee \ \text{Y-unsafe}$		
5.a		$\neg \left(\text{W-safe} \ \wedge \ \text{Y-unsafe}\right)$	Premise for subproof
5.b		$\text{W-unsafe} \ \wedge \ \text{Y-safe}$	CaseElim (left), lines _____ where $\varphi =$ _____ , and $\psi =$ _____
5.c		_____	_____
5.d		$\text{W-unsafe} \ \vee \ \text{Y-unsafe}$	_____
6	$\text{W-safe} \ \wedge \ \text{Y-unsafe} \ \vee \ \neg \left(\text{W-safe} \ \wedge \ \text{Y-unsafe}\right)$		
7	$\text{W-unsafe} \ \vee \ \text{Y-unsafe}$		

Exercise:

Problem:[Practice problem—solution provided.]



A sample WaterWorld board

Given the above figure, and using any of the immediately obvious facts as premises, prove that location P is safe by using our proof system and the WaterWorld axioms.

While this proof is longer (over two dozen steps), it's not too bad when sub-proofs are used appropriately. To make life easier, you may use the following theorem:
$\text{Q-has-1} \Rightarrow \text{P-safe} \wedge \text{R-safe} \vee \text{P-safe} \wedge \text{W-safe} \vee \text{R-safe} \wedge \text{W-safe}$, along with any proven previously. When looking at the given board, you can use premises like Y-safe as well as $\neg\text{Y-unsafe}$.

Solution:

1	Q-has-1		
2	X-has-1		
3	$\neg\text{Y-unsafe}$		
4	$\text{W-unsafe} \vee \text{Y-unsafe}$		
5	$\text{Y-unsafe} \vee \text{W-unsafe}$		

6	W-unsafe		
7	subproof:$\neg\neg$ (P-safe $\wedge$ W-safe) $\vdash$ false		
7.a		$\neg\neg$ (P-safe $\wedge$ W-safe)	Premise for subproof
7.b		P-safe $\wedge$ W-safe	$\neg$Elim, line 7.a
7.c		W-safe	$\wedge$Elim, line 7.b
7.d		W-safe $\Rightarrow$ $\neg$W-unsafe	WaterWorld axiom
7.e		$\neg$W-unsafe	$\Rightarrow$Elim, lines 7.c,7.d
7.f		false	falseIntro, lines 6,7.e
8	$\neg$ (P-safe $\wedge$ W-safe)		
9	subproof:$\neg\neg$ (R-safe $\wedge$ W-safe) $\vdash$ false		
9.a		$\neg\neg$ (R-safe $\wedge$ W-safe)	Premise for subproof
9.b		R-safe $\wedge$ W-safe	$\neg$Elim, line 9.a
9.c		W-safe	$\wedge$Elim, line 9.b
9.d		W-safe $\Rightarrow$ $\neg$W-unsafe	WaterWorld axiom
9.e		$\neg$W-unsafe	$\Rightarrow$Elim, lines 9.c,9.d
9.f		false	falseIntro, lines 6,9.e
10	$\neg$ (R-safe $\wedge$ W-safe)		
11	Q-has-1 $\Rightarrow$ P-safe $\wedge$ R-safe $\vee$ P-safe $\wedge$ W-safe $\vee$ R-safe $\wedge$ W-safe		
12	P-safe $\wedge$ R-safe $\vee$ P-safe $\wedge$ W-safe $\vee$ R-safe $\wedge$ W-safe		

13	$\text{R-safe} \wedge \text{W-safe} \vee \text{P-safe} \wedge \text{R-safe} \vee \text{P-safe} \wedge \text{W-safe}$		
14	$\text{P-safe} \wedge \text{R-safe} \vee \text{P-safe} \wedge \text{W-safe}$		
15	$\text{P-safe} \wedge \text{W-safe} \vee \text{P-safe} \wedge \text{R-safe}$		
16	$\text{P-safe} \wedge \text{R-safe}$		
17	P-safe		

Alternatively, the subproofs could easily have been pulled out into lemmas. Just like using subroutines in a program, that would make the proof somewhat clearer, even though in this case each lemma would be used only once.

Observe how the two subproofs have some identical lines (7.c-7.f and 9.c-9.f). It would be incorrect to replace those lines in the second subproof with a citation of the results of the first subproof. First, because the previous subproof had been completed, and moreover, the two subproofs have different premises. This is analogous to two subroutines that happen to have some identical code lines, even through they are called separately and have different parameters.

Note: Interestingly, we didn't need to use R-safe as a premise. (In fact, we nearly proved that ¬R-safe would have been inconsistent with the other premises.)

Exercise:

Problem:

Starting from the WaterWorld axiom
$\text{Q-has-1} \Rightarrow \text{P-safe} \wedge \text{R-safe} \wedge \text{W-unsafe} \vee \text{P-safe} \wedge \text{R-unsafe} \wedge \text{W-safe} \vee \text{P-unsafe} \wedge \text{R-safe}$ /
, we could prove the following theorem cited in the previous problem:
$\text{Q-has-1} \Rightarrow \text{P-safe} \wedge \text{R-safe} \vee \text{P-safe} \wedge \text{W-safe} \vee \text{R-safe} \wedge \text{W-safe}$.

Prove the following theorem which is slightly simpler: $\varphi \Rightarrow \psi \wedge \theta \vee \delta \wedge \varepsilon \vdash \varphi \Rightarrow \psi \vee \delta$.

Note: If you have trouble, first prove an even simpler version: $\varphi \Rightarrow \psi \wedge \theta \vdash \varphi \Rightarrow \psi$.

Exercise:

Problem:[Practice problem—solution provided.]

Show that the ¬Elim inference rule is redundant in our system. In other words, without using ¬Elim, prove that $\neg\neg\varphi \vdash \varphi$.

Solution:

1	$\neg\neg\varphi$			Premise
2	subproof:$\neg\varphi \vdash$ false			
2.a		$\neg\varphi$	Premise for subproof	
2.b		false	falseIntro, lines 1,2.a	
3	φ			RAA, line 2

Exercise:

Problem:

Show that the ¬Intro inference rule is redundant in our system. In other words, without using ¬Intro, prove that $\varphi \vdash \neg\neg\varphi$. To make sure that you're not hiding any uses of ¬Intro, also do not use any previous theorems.

Exercise:

Problem:

Show that the CaseElim inference rule is redundant in our system. For brevity, we'll just consider the left-hand version. In other words, without using CaseElim, prove that $\varphi \vee \psi, \neg\varphi \vdash \psi$. To make sure that you're not hiding any uses of CaseElim, also do not use any previous theorems.

Exercise:

Problem:

- State where on a board pirates could be positioned, so that: P-has-1 $\wedge$ U-has-1 $\wedge$ W-has-1, but X-safe.
- Compare this with a previous theorem, B-has-1 $\wedge$ G-has-1 $\wedge$ J-has-1 $\Rightarrow$ K-unsafe, the same idea shifted down a couple of rows. Suppose we try to translate this theorem's proof so as to conclude ¬X-safe (clearly untrue, by the above). What is the first step of the modified proof which doesn't hold when B,G,J,K are mindlessly replaced with P,U,W,X, respectively? (Just give a line number; no explanation needed. Your answer will be of the form "Lemma A line 1" or "main proof line 2".)
- We've just seen that the mindless changing of location-names introduces false steps. But we can be a little smarter, and modify the false step to get a formula which **is** true, and is also still in the spirit of the original proof. We can thus patch the problem from the previous part, and continue on modifying the original proof for several more steps. But clearly we can't translate the entire original proof; we eventually hit a more fundamental snag: a formula which isn't true, yet can't be patched up, either. What is the first line that can't be patched? (Again, just give a line number; no explanation needed. Your answer will be of the form "Lemma A line 1" or "main proof line 2".)

Exercise:

Problem:

Which is worse, having an unsound (but complete) inference system or an incomplete (but sound) one? Why?

Relations and Logic: using relations

Relations: Building a better (representation of) WaterWorld

So far, we have represented WaterWorld boards using propositions like A-has-2 and B-unsafe. You've probably already felt that this is unwieldy, having hundreds propositional variables running around, with only our naming convention implying any relation between them. Worse, this zoo of propositions doesn't reflect how we actually think about WaterWorld. For instance, the only way the rules recognize that locations A and B are near each other is because of several axioms which simultaneously involve A-has-2 and B-unsafe, etc., in just the right way to result in our idea of the concept "neighbor". In fact, there is no way of talking about the location A directly; we only had propositions which dealt with its properties, such as whether or not it neighbored exactly two pirates.

If writing a program about WaterWorld, our program should reflect our conception of the problem. However, as it stands, our conception corresponds to having many many Boolean variables named A-has-2, B-unsafe, etc. Even worse, the rules would be encodings of the hundreds of axioms. A long enumeration of the axioms is probably **not** how you think of the rules. In other words, when explaining the game to your friend, you probably say "if a location contains a 2, then two of its neighbors are pirates", rather than droning on for half an hour about how "if location A contains a 2, then either location B is unsafe or …".

Moreover, the original rules only pertained to a fixed-size board; inventing a new game played on a 50×50 grid would require a whole new set of rules! That is clearly **not** how we humans conceptualize the game! What we want, when discussing the rules, is a generic way to discussing neighboring locations, so that we can have one single rule, saying that if a (generic) location has a zero, then any neighboring location is safe. Thus, we allow the exact details of "neighboring location" to change from game to game as we play on different boards (just as which locations contain pirates changes from game to game).

In a program, you'd probably represent the board as a collection (matrix, list, whatever) of Booleans. In our logic, to correspond to this data structure, we'll introduce **binary relations**.

Note: By including relations (rather than sticking entirely with propositions), we are leaving the realm of propositional logic; we'll soon reach **first-order logic** once we also introduce quantifiers — corresponding to aspects of program control-flow (loops).

We'll start by adding a way to express whether any two locations are adjacent: a relation nhbr, which will encode the board's geography as follows: $\mathrm{nhbr}\,(A, B)$ and $\mathrm{nhbr}\,(Z, Y)$ are true, while $\mathrm{nhbr}\,(A, D)$ and $\mathrm{nhbr}\,(M, Z)$ are false.

What, exactly, do we mean by "relation"? We'll see momentarily, that we can represent nhbr as a set of pairs-of-locations (or equivalently, a function which takes in two locations, and returns either true or false.)

This relation "nhbr" entirely encodes the board's geography. Giving somebody the relation is every bit as good as to showing them a picture of the board (in some ways, better — the relation makes it perfectly clear whether two locations which just barely touch at a single point, like B and G, are meant to be considered neighbors.)

Exercise:

Problem:

We used a binary (two-input) relation to describe neighboring locations. How can we use a relation to capture the notion "location A is safe"?

Solution:

We'll use a **unary** (one-input) relation: $\mathrm{safe}(A)$ is true if and only if ("iff") location A is safe.

After defining relations and discussing their properties, we'll talk about interpreting logic formulas relative to particular relations.

Using relations gives us additional flexibility in modeling our domain, so that our formal logical model more closely corresponds to our intuition. Relations help separate the WaterWorld domain axioms (code) from the data, i.e., the particular board we're playing on.

properties of relations
Relations are subsets. Binary relations might be reflexive, symmetric, antisymmetric, or transitive.

When using relations in logic formulas, there are two things going on:

- relations themselves, as mathematical entities, and
- formulas involving symbols, which must be interpreted as specific relations.

First things first: we'll just discuss relations for now, and later tackle using relations in logic formulas.

We'll start with a couple of equivalent ways of defining relations, and then discuss a common subclass of relations: binary relations.

Relations as subsets

Consider the set of WaterWorld locations $\text{Loc} = \{A, B, \ldots, Z\}$. For this **domain** (also known as a **universe**), we'll say a **binary relation** is a set of (ordered) pairs of the domain.

Example:
For instance, the nhbr relation of the previous section is the set $\{(A, B), (A, G), (B, A), (B, C), \ldots, (Y, X), (Y, Z), (Z, Y)\}$.
That is, x is related to y if (x, y) is in the set nhbr.

Example:
For the domain $D = \{\text{Object}, \text{String}, \text{MutableString}\}$, the relation subclass-of might be $\{(\text{String}, \text{Object}), (\text{MutableString}, \text{Object}), (\text{MutableString}, \text{String})\}$.
In general, a binary relation over the domain D is a subset of $D \times D$. Note that these are **ordered** pairs; just because x is related to y doesn't mean y has the same relation to x. For example, while $(\text{MutableString}, \text{Object})$ is in the relation subclass-of, the pair $(\text{Object}, \text{MutableString})$ most certainly is not.

Example:
You can consider the relation hasStarredWith, over the domain of Hollywood actors. We won't list all the elements of the relation, but some related pairs are:

- $\text{hasStarredWith}\,(\text{Ewan McGregor}, \text{Cameron Diaz})$, as witnessed by the movie *A Life Less Ordinary*, 1997.
- $\text{hasStarredWith}\,(\text{Cameron Diaz}, \text{John Cusack})$, as witnessed by the movie *Being John Malkovich*, 1999.

If binary relations are subsets of pairs of the domain, what might a **unary** relation be? Simply, subsets of the domain.

Example:
For the domain of vegetables, Ian defines the relation yummy? as $\{\text{tomatoes}, \text{okra}, \text{cucumbers}, \text{carrots}, \text{potatoes}\}$ and nothing else.

Example:
In one particular game of WaterWorld, the relation hasPirate turned out to be $\{K, T, R, U, E\}$.

If unary and binary relations make sense, what about ternary, etc., relations? Sure! In general, a k-**ary** relation (or, "relation of **arity** k") over the domain D is a subset of D^k. However, any given relation has a fixed arity. That is, a relation may be binary or ternary, but not both.

As with propositions, rather than writing " $R(x, y)$ is true ", we'll simply write " $R(x, y)$ ". In fact, notice that once you choose some particular pair of x and y, then $R(x, y)$ can be treated as a single true/false proposition. (We'll soon extend the idea of propositions to include such relation symbols, and then allow formulas to include these **terms**.)

Example:
" $\text{prime}\,(18)$ " is a proposition that's false, assuming the standard interpretation of prime.

Example:
" $\text{safe}\,(A)$ " is a proposition that is true on some boards and not others.

Relations as functions

The relation nhbr, which we're defining as a set (of pairs), could also be thought of being a function. We say that the **indicator function** of a set is a Boolean function indicating whether its input is in the set or not. So instead of being given the set nhbr, you would have been equally happy with its indicator function f_{nhbr}, where (for example) $f_{\text{nhbr}}(B, C) = \text{true}$ and $f_{\text{nhbr}}(B, Q) = \text{false}$. Similarly, if you know that $f_{\text{hasPirate}}(K) = \text{true}$ and that $f_{\text{hasPirate}}(L) = \text{false}$, then this is enough information to conclude that $K \in \text{hasPirate}$ and $L \notin \text{hasPirate}$. **The set and the function are equivalent ways of modeling the same underlying relation.**

The next two exercises aren't meant to be difficult, but rather to illustrate that, while we've sketched these two approaches and suggested they are equivalent, we still need an exact definition.

Exercise:

Problem:

For the indicator function $f(x, y) = \begin{cases} \text{true} & \text{if } \ y = x^2 \\ \text{false} & \text{otherwise} \end{cases}$ on the domain of (pairs of) natural numbers, write down the set-of-pairs representation for the corresponding binary relation. It's insightful to give the answer both by listing the elements, possibly with ellipses, and also by using set-builder notation.

In general, for a binary indicator function f, what, exactly, is the corresponding set?

Solution:

$\{(0, 0), (1, 1), (2, 4), (3, 9), \ldots, (i, i^2), \ldots\}$ In set-builder notation, this is $\{(x, y) \ \ y = x^2\}$

In general, for an indicator function f, the corresponding set would be $\{(x, y) |\ f(x, y)\}$ (Note that we don't need to write " … $f(x, y) = \text{true}$ "; as computer scientists comfortable with Booleans as values, we see this is redundant.)

Exercise:

Problem:

For the relation $\text{hasPirate} = \{K, T, R, U, E\}$ on the set of (individual) WaterWorld locations, write down the indicator-function representation for the corresponding unary relation. **In general**, how would you write down this translation?

Solution:

$$f_{\text{hasPirate}}(x) = \begin{array}{ll} \text{true} & \text{if } x = K \\ \text{true} & \text{if } x = T \\ \text{true} & \text{if } x = R \\ \text{true} & \text{if } x = U \\ \text{true} & \text{if } x = E \\ \text{false} & \text{otherwise} \end{array} .$$

In general, for a (unary) relation R, $f_R(x) = \begin{cases} \text{true} & \text{if } x \in R \\ \text{false} & \text{if } x \notin R \end{cases}$.

Since these two formulations of a relation, sets and indicator functions, are so close, we'll often switch between them (a very slight abuse of terminology).

Think about when you write a program that uses the abstract data type `Set`. Its main operation is `elementOf`. When might you use an explicit enumeration to encode a set, and when an indicator function? Which would you use for the set of WaterWorld locations? Which for the set of prime numbers?

Functions as Relations

Some binary relations have a special property: each element of the domain occurs as the first item in exactly one tuple. For example,
$\text{isPlanet} = \{(\text{Earth}, \text{true}), (\text{Venus}, \text{true}), (\text{Sol}, \text{false}), (\text{Ceres}, \text{false}), (\text{Mars}, \text{true})\}$
is actually a (unary) function. On the other hand,
$\text{isTheSquareOf} = \{(0, 0), (1, 1), (1, -1), (4, 2), (4, -2), (9, 3), (9, -3), \ldots\}$ is not a function, for two reasons. First, some numbers occur as the first element of **multiple** pairs. Second, some numbers, like 3, occurs as the first element of **no** pairs.

We can generalize this to relations of higher arity, also. This is explored in this exercise and this one.

Binary Relations

One subclass of relations are common enough to merit some special discussion: binary relations. These are relations on pairs, like nhbr.

Binary Relation Notation

Although we introduced relations with prefix notation, e.g., $<(i, j)$, we'll use the more common infix notation, $i < j$, for well-known arithmetic binary relations.

Binary Relations as Graphs

In fact, binary relations are common enough that sometimes people use some entirely new vocabulary: A domain with a binary relation can be called **vertices** with **edges** between them. Together this is known as a **graph**. We won't stress these terms right now, as we're not studying graph theory.

Binary relations (graphs) can be depicted visually, by drawing the domain elements (vertices) as dots, and drawing arrows (edges) between related elements.

A binary relation with a whole website devoted to it is "has starred in a movie with". We'll call this relation $\mathrm{hasStarredWith}$ over the domain of actors. Some sample points in this relation:

- $\mathrm{hasStarredWith}\,(\mathrm{Ewan\ McGregor}, \mathrm{Cameron\ Diaz})$, as witnessed by the movie *A Life Less Ordinary*, 1997.
- $\mathrm{hasStarredWith}\,(\mathrm{Cameron\ Diaz}, \mathrm{John\ Cusack})$, as witnessed by the movie *Being John Malkovich*, 1999.

You can think of each actor being a "location", and two actors being "adjacent" to each other if they have ever starred in a movie together; two of these locations, even if not adjacent might have a multi-step path between them. (There is also a shorter path; can you think of it? The (in)famous Kevin Bacon game asks to find a shortest path from one location to the location Kevin Bacon. Make a guess, as to the longest shortest path leading from (some obscure) location to Kevin Bacon.)

Some other graphs:

- Vertices can be tasks, with edges meaning dependencies of what must be done first.

- In parallel processing, Vertices can be lines of code; there is an edge between two lines if they involve common variables. Finding subsets of vertices with no lines between them represent sets of instructions that can be executed in parallel (and thus assigned to different processors.)
- "Word ladders" seek to transform one word to another by changing one letter at a time, while always remaining a word. For example, a ladder leading from WHITE to SPINE in three steps is:

  - WHITE
  - WHINE
  - SHINE
  - SPINE

  If a solution to such a puzzle corresponds to a path, what do vertices represent? What are edges? Do you think there is a path from any 5-letter word to another?

interpretations
Interpretations map relation-symbols to an actual relation.

Needing Interpretations to Evaluate Formulas

You might have noticed something funny: we said $\text{safe}(a)$ depended on the board, but that $\text{prime}(18)$ was false. Why are some some relations different than others? To add to the puzzling, there was a caveat in some fine-print from the previous section: " $\text{prime}(18)$ is false **under the standard interpretation of prime** ". Why these weasel-words? Everybody knows what prime is, don't they? Well, if our domain is matrices of integers (instead of just integers), we might suddenly want a different idea "prime".

Consider the formula $E(x, x)$ true for all x in a domain? Well, it depends not only on the domain, but also on the specific binary relation E actually stands for:

- for the domain of integers where E is interpreted as "both are even numbers", $E(x, x)$ is false for some x.
- for the domain $\{2, 4, 6, 8\}$ where E is interpreted as "sum to an even number", $E(x, x)$ is true for every x.
- for the domain of integers where E is interpreted as "greater than", $E(x, x)$ is false for some x (indeed, it's false for **every** x).
- for the domain of people where E is interpreted as "is at least as tall as", $E(x, x)$ is true for every x.

Thus a formula's truth depends on the interpretation of the (syntactic, meaning-free) relation symbols in the formula.

Interpretation
: The interpretation of a formula is a domain, together with a mapping from the formula's relation symbols to specific relations on the domain.

One analogy is " Programs are to data, as formulas are to interpretations ". (In particular, the formula is a like a boolean function: it takes its input (interpretation), and returns true or false.)

Using Truth Tables to Summarize Interpretations (Optional)

Consider the formula $\phi = R(x, y) \Rightarrow S(x, y) \ \wedge \ \neg T(x, y)$. As yet, we haven't said anything about the interpretations of these three relations. But, we do know that each of $R(x, y)$, $S(x, y)$, and $T(x, y)$ can either be true or false. Thus, treating each of those as a proposition, we can describe the formula's truth under different interpretations.

$R(x, y)$	$S(x, y)$	$R(x, y)$	ϕ
false	false	false	true
false	false	true	true
false	true	false	true
false	true	true	true
true	false	false	false
true	false	true	false
true	true	false	true
true	true	true	false

Using Formulas to Classify Interpretations (Optional)

In the previous section, having a formula was rather useless until we had a particular interpretation for it. But we can view that same idea backwards:

Given a formula, what are all the interpretations for which the formula is true?

For instance, consider a formula expressing that an array is sorted ascendingly: For all numbers i,j, $(i < j) \Rightarrow (\text{element}\,(i) \leq \text{element}\,(j))$. But if we now broaden our mind about what relations/functions the symbols element, $<$, and $\leq$ represent and then wonder about the set of all structures/interpretations which make this formula true, we might find that our notion of sorting is broader than we first thought. Or equivalently, we might decide that the notion "ascending" applies to more structures than we first suspected.

Similarly, mathematicians create some formulas about functions being associative, having an identity element, and such, and then look at all structures which have those properties; this is how they define notions such as groups, rings, fields, and algebras.

Encoding Functions as Relations

What about adding functions, to our language, in addition to relations? Well, functions are just a way of relating input(s) to an output. For example, 3 and 9 are related by the square function, as are 9 and 81, and 0,0. Is any binary relation a function? No, for instance $\{(9, 81), (9, 17)\}$ is not a function, because there is no **unique** output related to the input 9.

How can we enforce uniqueness? The following sentence asserts that for each element x of the domain, R associates at most one value with x: For all x, y and z of the domain,
Equation:

$$R(x, y) \ \wedge \ R(x, z) \Rightarrow (y = z)$$

This is a common trick, for to describe uniqueness: if y and z each have some property, then they must be equal. (We have not yet specified that for

every element of the domain, there is **at least** one element associated with it; we'll get to that later.)

Exercise:

Problem:

We just used a binary relation to model a unary function. Carry on this idea, by using a ternary relation to start to model a binary function. In particular, write a formula stating that for every pair of elements w, x in the domain, the relation S associates at most one value with that pair.

Solution:

For all w, x, y, and z of the domain,

Equation:

$$S(w, x, y) \ \wedge \ S(w, x, z) \Rightarrow (y = z)$$

Relations and Logic: Non-standard Interpretations

Prime factorization

Note that there are other possible interpretations of “prime”. For example, since one can multiply integer matrices, there might be a useful concept of “prime matrices”.

For example: Consider **only** the numbers $F = \{1, 5, 9, 13, \ldots\}$ — that is, $F = \{k, 4k+1 | \, k \in \mathbb{N}\}$. It's easy to verify that multiplying two of these numbers still results in a number of the form $4k+1$. Thus it makes sense to talk of factoring such numbers: We'd say that 45 factors into $5 9$, but 9 is considered prime since it doesn't factor into smaller elements of F.

Interestingly, within F, we lose **unique** factorization:
$441 = 9 \times 49 = 21 \times 21$, where each of 9, 21, and 49 are prime, relative to F! (Mathematicians will then go and look for exactly what property of a multiplication function are needed, to guarantee unique factorization.)

The point is, that all relations in logical formula need to be interpreted. Usually, for numbers, we use a standard interpretation, but one can consider those formulas in different, non-standard interpretations!

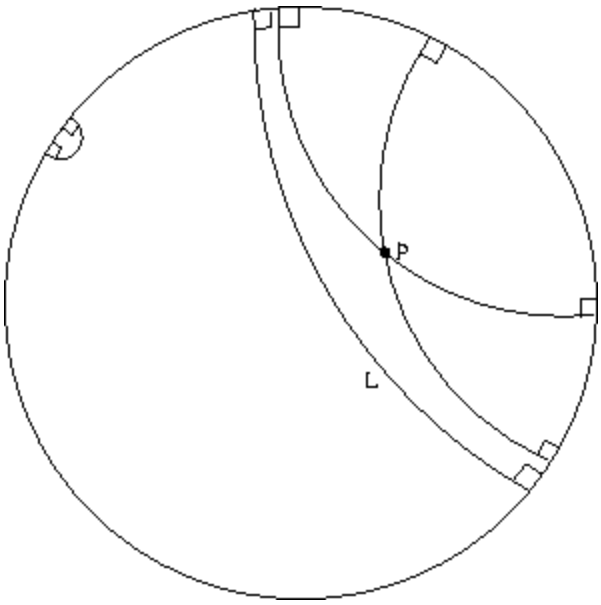
The Poincaré Disc

A long outstanding problem was that of Euclid's parallel postulate: Given a line and a point not on the line, how many lines parallel to the first go through that point? Euclid took this as an axiom (unable to prove that it followed from his other axioms). Non-Euclidean geometries of Lobachevsky and Riemann took different postulates, and got different geometries. However, it was not clear whether these geometries were **sound** — whether one could derive two different results that were inconsistent with each other.

Henri Poincaré developed an ingenious method for showing that certain non-Euclidean geometries **are** consistent — or at least, as consistent as Euclidean geometry. Remember that in Euclidean geometry, the concepts

“point” and “line” are left undefined, and axioms are built on top of them (e.g., “two different lines have at most one point in common”). While it's usually left to common sense to interpret “point”, “line”, and “a point is on a line”, any interpretation which satisfies the axioms means that all theorems of geometry will hold.

The Poincaré disc is one such interpretation: “point” is taken to mean “a point in the interior of the unit disc”, and “line” is taken to mean “a circular arc which meets the unit disc at right angles”. So a statement like “two points determine a line” can be interpreted as "[*] For any two points inside the disc, there is exactly one circular arc which meets the disc at right angles." Indeed, this interpretation preserves all of Euclid's postulates **except** for the parallel postulate. You can see that for a given line and a point not on it, there are an infinite number of parallel (that is, non-intersecting) lines.



Some lines in the Poincaré disc, including several lines parallel to a line L through a point p.

(Note that the distance function is very different within the Poincaré disc; in fact the perimeter of the disc is off at infinity. Angles, however, do happen to be preserved.)

The critical point of his interpretation of a non-Euclidean geometry is this: **it is embedded in Euclidean geometry!** So we are able to prove (within the embedding Euclidean geometry) that the disc-postulates hold (e.g., we can prove the statement [*] above as a theorem about circular arcs in Euclidean geometry). Therefore, if there is any inconsistency in non-Euclidean geometry, then that could be parlayed into some inconsistency of Euclidean geometry. Thus, his interpretation gives a proof that the strange non-Euclidean geometry is as sound as our familiar Euclidean geometry.

P vs. NP and Oracles

A well-known problem in computer science — “P vs. NP” — asks whether (for a given problem) it is truly more difficult to find a short solution (when one exists) (“NP”), than it is to verify a short purported solution handed to you (“P”). For example, “Given a set of people and how strong person is, can you partition them into two tug-of-war teams which are exactly evenly matched?” Certainly it **seems** easier to check that a pair of proposed rosters has equal strength (and, verify that everybody really is on one team or the other) than to have to come up with two perfectly-matched teams. But conceivably, the two tasks might be equally-difficult up to some acceptable (polynomial time) overhead. While every assumes that P is easier than NP, nobody has been able to prove it.

An interesting variant of the problem lets both the problem-solver and the purported-answer-verifier each have access to a particular **oracle** — a program that will gives instant yes/no answers **to some other** problem (say, “given any set of numbers, yes or no: is there an even-sized subset whose total is exactly the same as some odd sized subset?”).

It **has** been shown that there is some oracle which makes the problem-solver's job provably tougher than the proof-verifier's job, and also there is some other oracle problem-solver's job provably no-tougher than the proof-verifier's job.

This means that any proof of P being different from NP has to be subtle enough so that when P and NP are re-interpreted as “P and NP with respect to a particular oracle”, the proof will no longer go through. Unfortunately, this eliminates all the routine methods of proof; we know that solving this problem will take some new attack.

Löwenheim-Skolem and the real numbers

The Löwenheim-Skolem theorem of logic states that if a set of (countable) domain axioms has a model at all, then it has a **countable** model. This is a bit surprising when applied to the axioms of arithmetic for the real numbers: even though the real numbers are uncountable, there is some **countable** model which meets all our (finite) axioms of the real numbers!

Object-oriented programming

Note that object-oriented programming is founded on the possibility for nonstandard interpretations: perhaps you have some code which is given a list of `Object`s, and you proceed to call the method `toString` on each of them. Certainly there is a standard interpretation for the function `Object.toString`, but your code is built to work even when you call this function and some nonstandard, custom, overridden method is called instead.

It can become very difficult to reason about programs when the run-time method invoked might be different from the one being called. We're used to specifying **type** constratins which any interpretation must satisfy; wouldn't it be nice to specify more complicated constraints, e.g. “this function returns an `int` which is a valid index into [some array]”? And if we can describe the constraint **formally** (rather than in English comments, which is how most code works), then we could have the computer enforce that contract! (for every interpretation which gets executed, including non-static ones).

An obvious formal specification language is code itself — have code which verifies pre-conditions before calling a function, and then runs code verifying the post-condition before leaving the function. Indeed, there are

several such tools about (Java, Scheme). In the presence of inheritance, it's harder than you might initially think to do this correctly.

It is still a research goal to be able to (sometimes) optimize away such run-time verifications; this requires **proving** that some code is correct (at least, with respect to its post-condition). The fact that the code might call a function which will be later overridden (our “non-standard interpretations”) exacerbates this difficulty. (And proving correctness in the presence of concurrency is even tougher!)

Even if not proving programs correct, being able to specify contracts in a formal language (code or logic) is a valuable skill.

Real-World Arguments

Finally, it is worth noting that many rebuttles of real world arguments (see also some exercises) amount to showing that the argument's form can't be valid since it doesn't hold under other interpretations, and thus there must be some unstated assumptions in the original.

modeling with relations
(Blank Abstract)

Modeling with Relations

Note: Note that the nhbr relation can actually represent an arbitrarily weird board, such as locations that look adjacent on the map but actually aren't, boards which wrap around a cylinder or toroid, or a location with a tunnel connecting it to a location far across the board (like the secret passages in the game Clue, or the harrowing sub trip through Naboo in *Star Wars: The Phantom Menace*.) One-way passages can be encoded as well (meaning the nhbr relation need not be symmetric). Actually, **any** graph can be represented!

Exercise:

Problem:

How shall we encode concepts such as " location A has 3 dangerous neighbors ", using relations?

Solution:

A good first guess might be to say we have a function which returns the number of pirates next to a given location. That is, " $\mathrm{piratesNear}\,(A) = 3$ ". However, "$\mathrm{piratesNear}$" doesn't qualify as a relation. Why not?

To work around this, we could propose a binary relation along the lines of " $\mathrm{piratesNear}\,(A, 3) = \mathrm{true}$ ". This is better, but it requires our domain to be not only board locations, but also numbers. And to be able to talk about numbers, we'd need more axioms, as well as numeric relations such as >.

Note: Hmmm, what is the arity of ">"?

> While this approach is feasible, and ultimately might be what we want, for now, let's stick with relations involving only locations, not numbers.
>
> Okay, the third time's the charm: we'll implement the concept "A neighbors three pirates" as a relation $\text{has-3}(A)$ being true. To cover the cases when there are exactly two neighboring pirates, we'll use a whole new separate relation, "has-2"; $\text{has-2}(A)$ would be false on any board where $\text{has-3}(A)$ is true (at least, in our standard interpretation).

Proofs otherwise unchanged. Note that we might express our rules as " for any locations x and y, we have the following axiom:
$\text{has-3}(x) \ \wedge \ \text{nhbr}\,(x, y) \Rightarrow \neg\,\text{safe}\,(y)$ ". Really, note that there's something else going on here: x and y are symbols which can represent **any** location: they are variables, whose value can be any element of the domain.

For the domain of types-of-vegetables, the relation yummy is a useful one to know, when cooking. In case you weren't sure,
$\text{yummy}\,(\text{Brussels sprouts}) = \text{false}$, and $\text{yummy}\,(\text{carrots}) = \text{true}$.

Suppose we had a second relation, yucky. Is it conceivable that we could model a vegetable that's neither yucky nor yummy, using these relations? Sure! (Iceberg lettuce, perhaps.) In fact, we could even have a vegetable which is both yummy and yucky — radishes!

Note: A quick digression on a philosophical nuance: the domain for the above problem is **not** vegetables; it's types-of-vegetables. That is, we talk about whether or not carrots are yummy; this is different than talking the yumminess of the carrot I dropped under the couch yesterday, or the carrot underneath the chocolate sauce. In computer science, this often manifests itself as the difference between values, and types of values. As examples, we distinguish between 3 and the set of all integers, and we distinguish between particular carrots and the abstract idea of carrots. (Some languages even include types as values.) Philosophers enjoy debating how particular instances define the abstract generalization, but for our purposes we'll take each both vegetables and types-of-vegetables as given.

Exercise:

Problem:

You might have objected to the idea of the unary relation yummy, since different people have different tastes. How could you model individuals' tastes? (Hint: Use a **binary** relation.))

Solution:

We can use the binary relation $\mathrm{thinksIsYummy}$: In particular, $\mathrm{thinksIsYummy}\,(\mathrm{Ian}, \mathrm{anchovies}) = \mathrm{false}$ but $\mathrm{thinksIsYummy}\,(\mathrm{Phokion}, \mathrm{anchovies}) = \mathrm{true}$ What set are we using, as the domain for this? Really, the domain is the union of people and pizza-toppings. So $\mathrm{thinksIsYummy}\,(\mathrm{radishes}, \mathrm{brusselsSprouts})$ is a valid thing to write down; it would be false. Note that if working with such a domain, having unary predicates $\mathrm{isVegetable}$ and $\mathrm{isPerson}$ would be useful.

Modeling actors and the has-starred-with relation didn't include information about specific movies. For instance, it was impossible to write any formula which could capture the notion of three actors all being in the same movie.

Exercise:

Problem:

Why doesn't $\mathrm{hasStarredWith}\,(a, b) \wedge \mathrm{hasStarredWith}\,(b, c) \wedge \mathrm{hasStarredWith}\,(c, a)$ capture the notion of a, b, and c all being in the same movie? Prove your answer by giving a counterexample.

Solution:

The proposed formula asserts that each pair has been in some movie together, but they each could have been different movies without being in the same one simultaneously. As a counterexample, it **is** true that $\mathrm{hasStarredWith}\,(\mathrm{Charlie\ Chaplin}, \mathrm{Norman\ Lloyd})$ (as witnessed by *Limelight*, 1952), $\mathrm{hasStarredWith}\,(\mathrm{Norman\ Lloyd}, \mathrm{Janeane\ Garofolo})$ (as witnessed by *The Adventures of Rocky and Bullwinkle*, 2000), and if we generously include archive footage, $\mathrm{hasStarredWith}\,(\mathrm{Charlie\ Chaplin}, \mathrm{Janeane\ Garofolo})$ (as witnessed by *Outlaw Comic: The Censoring of Bill Hicks*, 2003); however, they have not

all been in a movie together. Might the counterexample you chose become nullified, in the future?

Exercise:

Problem:

How might we make a model which **does** capture this? What is the domain? What relations do you want?

Solution:

As always, there are several ways of modeling this problem. We'll outline three.

First, we could augment the hasStarredWith to be a ternary (3-input) relation to include the movie. Like in the yummy extension, the domain would then include both actors and movies, and we'd also want relations to know which is which.

Second, we could use a bunch of relations. Starting with the familiar binary hasStarredWith, we'd add the ternary hasStarredWith3, the quaternary hasStarredWith4, …. Our domain would just be actors. However, we'd either need an infinite number of such relations, which we normally don't allow, or we'd need an arbitrary cap on the number of people we're interested in at a time.

Third, we could use **sets** of actors, instead of individuals. We'd need only one relation, haveStarredtogether, that states a set of actors have starred together in a single movie.

Of course, the notion of interpretations are still with us, though usually everybody wants to be thinking of one standard interpretation. Consider a relation with elements such as $\text{isChildOf}(\text{Bart}, \text{Homer}, \text{Marge})$. Would the triple $(\text{Bart}, \text{Marge}, \text{Homer})$ be in the relation as well as $(\text{Bart}, \text{Homer}, \text{Marge})$?

As long as all the writers and users of formulas involving isChildOf all agree on what the intended interpretation is, either convention can be used.

A Case Study: iTunes

Consider iTunes' "smart playlists": you can create a playlist consisting of (say) " All songs I've rated 3-stars or better, and whose genre is not Classical ". This is "smart" because its a program which is re-run every time your music library changes: For example, if you change a song's genre, it may be immediately added or deleted from the playlist. We realize actually have a simple formula (which we can express in propositional logic with relations). The structure (instance) for a single is the interpretation. This formula is true when interpreted on (my library's representation of) Brian Eno's "Here Come the Warm Jets", but false for Bonnie Tyler's '80's epic "Holding Out for a Hero" and for Bach's " 'Little' Fugue in Gm ". We now have one formula, and want to determine its truth-value in many different particular interpretations. In fact, we want to return all interpretations which make the formula (playlist) true.

Exercise:

Problem:

Look at the GUI box for defining these queries. Compared to propositional logic, what sort of formulas can you define?

Solution:

This is effectively Disjunctive or Conjunctive Normal Form, limited to clauses of one term each.

Exercise:

Problem:

Are there queries which can't be directly transliterated into the GUI box? [footnote]
"Transliterate" meaning a word-for-word substitution, while "translate" preserves meanings and idioms. So while the German "Übung macht den Meister" transliterates to "Drill makes the master", it translates to "Practice makes perfect".

Solution:

Yes. Two examples are $\neg\left((\text{genre} = \text{Classical}) \ \vee \ (\text{genre} = \text{Holiday})\right)$, and $(\text{genre} = \text{Rock}) \ \wedge \ ((\text{Rating} \geq 4) \ \vee \ (\text{genre} = \text{Classical}))$.

Exercise:

Problem:

Can you find formulas equivalent to each of the preceding two, which can be expressed?

Solution:

For the first example, $\neg\left((\text{genre} = \text{Classical}) \ \vee \ (\text{genre} = \text{Holiday})\right)$, we can clearly use DeMorgan's law and make the query
$\neg\left(\text{genre} = \text{Classical}\right) \ \wedge \ \neg\left(\text{genre} = \text{Holiday}\right)$.

However, for
$(\text{genre} = \text{Rock}) \ \wedge \ ((\text{Rating} \geq 4) \ \vee \ (\text{genre} = \text{Classical}))$ there is no equivalent one-term-per-clause DNF or CNF formula! [footnote]
Budding logicians might wonder how you actually **prove** this claim!

Fortunately, iTunes has a way around this. Playlist membership or non-membership is itself an available predicate, allowing you to nest playlists. Thus, you can build a playlist GoodOrClassical for
$(\text{Rating} \geq 4) \ \vee \ (\text{genre} = \text{Classical})$, then another
$(\text{genre} = \text{Rock}) \ \wedge \ \text{GoodOrClassical}$ for the desired result.

The upshot is that iTunes came up with a query language which is as expressive as propositional[footnote] logic. For some queries, it can be awkward to use, but the GUI designers who came up with smart playlists might have figured that few users would want such queries.

Note: How might you create a GUI widget which can specify **any** propositional formula, and yet still look nice and be intuitive enough for my mother to use? Is there a better usability/expressibility trade-off than what iTunes has done, or are they optimal?

Technically, this is a "relational calculus" formula, since we are using relations instead of flat propositions.

syntax and semantics of quantifiers
Introducting quantifiers, to upgrade from propositional logic to first-order logic.

Talking about unnamed items

Suppose we want to express a statement like "there is a location which has two neighbors" (which is true, at least for the domain of WaterWorld board locations), or " all actors have co-starred with Kevin Bacon " (which isn't true, at least for the domain of all Hollywood actors). As it stands, we can formulate these only awkwardly, by talking about **specific** (constant) locations like A and G, or specific actors like Ewan McGregor and Cameron Diaz. To talk about all locations, or actors, we're forced to make huge formulas such as $\text{nhbr}(Z, Y) \ \wedge \ \neg \text{nhbr}(Z, A) \ \wedge \ \neg \text{nhbr}(Z, B) \ \wedge \ \ldots \ \wedge \ \neg \text{nhbr}(Z, X)$, just to express "there is a location which has only one neighbor".

We'll redress this by introducing two **quantifiers**, ∃ ("there exists") and ∀ ("for all"). For example, "all actors have co-starred with Kevin Bacon" will be written $\forall a : (\text{coStarredWith}(a, \text{Kevin Bacon}))$. For " there is a location which has (at least) two neighbors ", we'll start with " there exists a location x … ", written $\exists x : (\ldots)$.

"For all" is really just an abbreviation for a large conjunction, while "exists" is a disjunction (it could also be called "for some", though it's not). How large a conjunction/disjunction? As big as your domain, which actually could be very small, or it could be infinitely large. Even aside from the fact that we can't write down an infinitely large conjunction or disjunction, quantifiers let us form the conjunction without having to select a domain in advance.

To continue with our WaterWorld example, how can we express the concept "x has (at least) two neighbors"? Well, we'll rephrase this as, " there exist distinct locations, y and z, which each of which is a neighbor of x ", written $\exists x : (\exists y : (\exists z : ((y \neq z) \ \wedge \ \text{nhbr}(x, y) \ \wedge \ \text{nhbr}(x, z))))$. We need the condition $\neg (y = z)$ in that formula to ensure that we have distinct locations. Compare to the algebraic equation $x + y = 4$ in which one possible solution is $x = y = 2$. Variables act the same way in both logic and algebra: different variables can happen to take on the same value.

We use quantifiers all the time in natural language. Consider the following examples, where we provide a natural English wording together with an equivalent phrasing that makes the quantification more explicit. We'll take the translations with a grain of salt, since sometimes people can disagree on the exact details of the intended English meaning. Such ambiguity can sometimes be a rich source of creativity, but it's not tolerable when documenting safety properties of software. While some of these examples are a bit frivolous, in general quantifiers let us precisely capture more interesting concepts in type-checking, data structures such as trees and hash tables, circuit specifications, etc.

Natural English	**Formalized English**
" If you don't love yourself, you can't love anybody else. "	" If you don't love you, there does not exists a person y, such that you love y. "
"N*Sync is the best band ever!"	" For all bands x, N*Sync is better than band x (or, $x =$ N*Sync). " A quick listen can easily show this statement false.
A casually subtle line from *Something About Mary*: "Every day is better than the next."	" For all days x, x is better than next(x). "

Natural English	Formalized English
A buggy line from a song (*Everybody Loves My Baby*, Jack Palmer and Spencer Willson, 1924): "Everybody loves my baby; My baby don't love [anybody] but me."	" For all persons x, x loves my baby. For all persons y, if my baby loves y, then y is me. " If true, one can conclude the speaker is his own baby, and is narcissistic.
"Every neighbor of x is unsafe."	"For all locations y, if y is a neighbor of x, then y is unsafe."
"There is a safe location that is a neighbor of x, if num(x)<3."	"If num(x)<3, then there exists a location y, such that y is safe, and y is a neighbor of x."
"If you've seen one episode, you've seen 'em all."	"If there exists one episode x such that you've seen x, then for all episodes z, you've seen z."
"Somebody loves everybody."	"There exists some person y, such that for all persons x, y loves x."
"There is someone for everybody."	"For all persons x, there exists a person y, such that y is for x."
"All's well that ends well."	"For all events x, if x ends well then x is well."

Quantification in English

Warning: The Ambiguous "Any"

The ambiguous "any": I was playing a game with some friends, and we came across the rule: " If you have more cards than any other player, then discard a card. " Does this mean "than **all** other players", or "than **some** other player"? Our group's opinion was divided (incl. across many native English speakers).

In our class terms, it's not always clear whether "any" means for-all, or for-some (there-exists). Or maybe more accurately, in the phrase "for any x", does x necessarily mean an arbitrary player?

Note: Linguistics students, or those who are so sure the rule clearly intended "than all other players": Switching " $x > y$ " to " $x < y$ " changes from an active voice to a passive voice but may also reverse your interpretation of the English quantifier "any": "If any player has fewer points than you, …"

In your proof-writing (and your English writing, and your informal writing), think about replacing "any" with either "every" or with "some", to make your meaning clear.

First-order logic: WFFs revisited

We originally defined a well-formed formula (WFF) for propositional logic; we'll extend this to WFFs for **first-order logic**, also known as **predicate logic**. At the same time, we'll more precisely define the binding of variables.

This logic allows use of both functions and relations. Since these functions' outputs are not Booleans (otherwise, we'd call them relations), but rather data than can be used as a relation's input, we separate the syntax into that of **terms** and formulas. Terms are all the possible inputs for a relation.

term
- A variable.
- A constant.
- A function applied to one or more terms.

Example:
$a, b, \ldots$

Example:
WaterWorld location F, Kevin Bacon, or the number 3.

Example:
$\text{successor}\ (3)$

Well-Formed Formula (WFF) for first-order logic
- A constant: $\texttt{true}$ or $\texttt{false}$.
- An **atomic formula**: a relation symbol applied to one or more terms.
- A negation of a WFF, $\neg\varphi$.
- A conjunction of WFFs, $\varphi \ \wedge \ \psi$.
- A disjunction of WFFs, $\varphi \ \vee \ \psi$.
- An implication of WFFs, $\varphi \Rightarrow \psi$.
- A **universal quantification** of a WFF, $\forall x : (\varphi)$.
- An **existential quantification** of a WFF, $\exists x : (\varphi)$.

Example:
$\text{nhbr}\ (x, F)$

Example:
$\forall x : (\text{nhbr}\ (x, F))$

Example:
$\exists x : (\text{nhbr}\ (x, F))$

While a formula is just a piece of syntax, the meaning of its connectives, including the quantifiers, is part of the definition of a WFF. However, as previously discussed, the meaning of a WFF also depends on the interpretation we give to its relations.

Examples

Example:
Everybody likes John Cusack: $\forall x : (\text{likes}\,(x, \text{John Cusack}))$.

Example:
Somebody likes Joan Cusack: $\exists x : (\text{likes}\,(x, \text{Joan Cusack}))$.

Example:
Somebody likes everybody: $\exists x : (\forall y : (\text{likes}\,(x, y)))$. (We use n for "needy"?)

Example:
Everybody likes somebody: $\forall y : (\exists x : (\text{likes}\,(y, x)))$. Careful; this formula looks similar to the preceding one, but it has a very different meaning!

Exercise:

Problem: How would you express "Somebody is liked by everybody"?

Solution:

The cue "Somebody …" suggests one person who exists; we'll call them p for "popular": $\exists p : (\ldots)$. Now we need to fill in the dots with "everybody likes p", to get: $\exists p : (\forall x : (\text{likes}\,(x, p)))$.

Exercise:

Problem: How would you express "Everybody is liked by somebody"?

Solution:

The cue "Everybody …" suggests a universal; we'll call them j for "J. Doe": $\forall j : (\ldots)$. Now we need to fill in the dots with "somebody likes j", to get: $\forall j : (\exists x : (\text{likes}\,(x, j)))$. Note that this formula is just like the preceding "Somebody likes everybody" example, except that the quantifiers have been swapped (and different variable names were used, a superficial difference).

Example:
The following formula is a simple application of symmetry.
$\forall x : (\forall y : (\text{near}\,(x, y) \Rightarrow \text{near}\,(y, x))) \ \wedge \ \text{near}\,(\text{Sue}, \text{Joe}) \Rightarrow \text{near}\,(\text{Joe}, \text{Sue})$.
While it is certainly true under the intended interpretation, it is also true under **any** interpretation. Such formulas are called **valid**. Valid first-order formulas are the natural analog of tautological propositional formulas.

Example:
$\forall x : (\text{even}\,(x) \ \wedge \ \text{prime}\,(x) \Rightarrow (x = 2))$ is a mathematical fact, in the standard interpretation of arithmetic.

While technically not allowed by our term and formula syntax, we'll continue using infix notation for common mathematical functions and relations, as in the previous example.

Exercise:

Problem:

The previous example used the relations even and prime. Of course, to use such relations, they must either be defined directly by the interpretation, or be defined in terms of functions and relations provided by the interpretation.

How would you define these two relations in terms of the basic numerical functions (addition, multiplication, …) and relations (= , <, >)?

Solution:

"Evenness" is a straightforward translation from " An integer n is even, iff it is twice some other integer k ": $\text{even}\,(n) \equiv \exists k : (n = 2k)$. Note that by this standard definition, zero is even.

There are many equivalent ways to define primality, just as there many algorithms for checking primality. One straightforward solution is $\text{noncomposite}\,(n) \equiv \forall j : (\forall k : ((jk = n) \Rightarrow (j = 1) \ \vee \ (k = 1)))$. Well, this is almost expresses "prime", except that $n = 1$ satisfies this formula. A mathematician points out that just as 0 is neither positive nor negative, 1 is neither prime nor composite; as stated this formula actually captures "noncomposite", oops. There are several ways to upgrade this to exactly capture "prime".

Note: 1 is called a "unit". If we consider the domain of all integers (not just natural numbers), the idea of primality still makes sense; -17 is also prime; and -1 is also another unit. Similarly, considering the domain of "complex integers" $\{a, b, a + bi | \ a \in \mathbb{Z} \ \wedge \ b \in \mathbb{Z}\}$ (could be written " $\mathbb{Z} + \mathbb{Z}i$ "), then i and $-i$ are also units. How might we generalize our definition of prime, to work in these further interpretations?

A similar, equivalent formula to the above is
$\text{noncomposite}\,(n) \equiv \neg\exists j : (\exists k : ((jk = n) \ \wedge \ (j \neq 1) \ \wedge \ (k \neq 1)))$.

Exercise:

Problem:

One hypothesis about natural numbers is known as Goldbach's Conjecture. It states that all even integers greater than two can be expressed as the sum of two primes. It is one of the oldest still-unsolved problems about numbers. How would you write this conjecture as a WFF?

Solution:

$\forall n : (\text{even}\,(n) \ \wedge \ (n > 2) \Rightarrow \exists p : (\exists q : (\text{prime}\,(p) \ \wedge \ \text{prime}\,(q) \ \wedge \ (p + q = n))))$

Enough about number theory. Let's look at some examples about common data structures and some about our favorite problem, WaterWorld.

Example:
If your program uses binary search trees and your domain is tree nodes, you need to know
$\forall \text{node} : ((\text{data}\,(\text{left}\,(\text{node})) \leq \text{data}\,(\text{node})) \ \wedge \ (\text{data}\,(\text{right}\,(\text{node})) > \text{data}\,(\text{node})))$. If these trees are also balanced, you need to know

$\forall \text{node} : ((\text{height}(\text{left}(\text{node})) = \text{height}(\text{right}(\text{node}))) \ \vee \ (\text{height}(\text{left}(\text{node})) + 1 = \text{height}(\text{right}(\text{node})$
. Again, these assume the implied interpretations.

Example:
We would like to be able to state that the output of a sorting routine is, in fact, sorted. Let's assume we're sorting arrays into ascending order.
To talk about the elements of an array in a typical programming language, we would write something like $a[i]$.
For this example, we'll use that notation, even though it doesn't quite fit the logic's syntax.
To describe sortedness (in non-decreasing order), we want to state that each element is greater than or equal to the previous one. However, just like in a program, we need to ensure our formula doesn't index outside the bounds of the array. For this example, we'll assume that an array's indices are zero to (but not including) size (a).
$\text{sorted}(a) \equiv \forall i : ((1 \leq i) \ \wedge \ (i < \text{size}(a)) \Rightarrow (a[i-1] < a[i]))$
When proving things about programs, it's often useful to realize there are alternate ways of defining things. So, let's see a couple more definitions.
We could change our indexing slightly: $\text{sorted}(a) \equiv \forall i : ((0 \leq i) \ \wedge \ (i < \text{size}(a) - 1) \Rightarrow (a[i] < a[i+1]))$.
Or we could state that the ordering holds on every pair of elements:
$\text{sorted}(a) \equiv \forall i : (\forall j : ((0 \leq i) \ \wedge \ (i < \text{size}(a)) \ \wedge \ (0 \leq j) \ \wedge \ (j < \text{size}(a)) \ \wedge \ (i < j) \Rightarrow (a[i] \leq a[j])))$
. This definition isn't any stronger, but it makes an additional property explicit. Generally, you'd find it harder to prove that this formula was true, but once you did, you'd find it easier to use this formula to prove other statements.

Exercise:

Problem:

The two preceding examples used functions like left, size, and subtraction, although our logic syntax doesn't include such functions. However, we can rewrite any use of functions with appropriate new relations.

As an example, rewrite $i < \text{size}(a)$ in proper first-order syntax.

Solution:

We need a new relation that combines the syntax of $<$ and size. The result would look like $\text{less-than-size}(i, a)$. This assumes the new relation has the obvious intended definition.

Exercise:

Problem:

One simple WaterWorld fact is that if a location has no unsafe neighbors, then its number of adjacent pirates is zero. Furthermore, the implication goes both ways. How would you state that as a WFF?

Solution:

$\forall x : (\forall y : (\text{nhbr}(x, y) \Rightarrow \text{safe}(y)) \Leftrightarrow \text{has-0}(x))$

Exercise:

Problem: How would you make a similar statement about the number of adjacent pirates being one?

Solution:

There are various solutions, but they all must capture the same idea: there exists exactly one unsafe neighbor. This solution states that in two parts:

- There exists an unsafe neighbor, u.
- Every unsafe neighbor is u.

Together, these two parts imply there is only one such u.

$$\forall x : (\exists u : (\text{nhbr}\,(x, u)\ \wedge\ \neg\,\text{safe}\,(u)\ \wedge\ \forall y : (\text{nhbr}\,(x, y) \Rightarrow (\neg\,\text{safe}\,(y) \Leftrightarrow y = u))) \Leftrightarrow \text{has-1}(x))$$

These statements are very similar to, and provable from, the first-order WaterWorld domain axioms.

A hint on deciphering formulas' meanings

Some formulas can get pretty hairy: $\forall x : (\exists y : (\forall z : (\text{likes}\,(x, y)\ \wedge\ \neg\,\text{likes}\,(y, z))))$. The zeroth step is to take a breath, and read this in English: for every x, there's some y such that for every z, x likes y but y doesn't like z. Even so, how do we approach getting a handle on what this means? Given an interpretation, how do we know it's true?

The top-down way would be to read this formula left-to-right. Is the whole formula true? Well, it's only true if, for every possible value of x, some smaller formula is true (namely, " there exists a y such that forall z, likes (x, y) and $\neg$ likes (y, z). "). (This is a formula with x free, that is, it's a statement about x.) And is that formula true? Well, precisely when we can find some y such that … (and so on). This direct approach is hard to keep inside your head all at once.

Most people prefer approaching the problems in a bottom-up manner (or if you prefer, right-to-left or inside-out): First consider at the small inner bits alone, figure out what they mean, and only then figure out how they relate.

- What does the innermost formula likes $(x, y)\ \wedge\ \neg$ likes (y, z) mean, in English? That's not so bad: x likes y, and y dislikes z. A statement about three people called x, y, z.
- Working outward, what does $\forall z : (\text{likes}\,(x, y)\ \wedge\ \neg\,\text{likes}\,(y, z))$ mean? Ah, not so bad either: x likes y, and y dislikes everybody. [footnote]
  Or if you prefer, "x likes y, who is a misanthrope". A self-loathing misanthrope, at that!
- Keep on going: $\exists y : (\forall z : (\text{likes}\,(x, y)\ \wedge\ \neg\,\text{likes}\,(y, z)))$ becomes "x likes some misanthrope".
- Now it's clear: $\forall x : (\exists y : (\forall z : (\text{likes}\,(x, y)\ \wedge\ \neg\,\text{likes}\,(y, z))))$ is just "everybody likes some misanthrope".

Phew!

"Forall"'s friend "if"

We have already seen quite a few formulas of the general form $\forall x : (P(x) \Rightarrow \ldots)$. Indeed, this is a very useful idiom: If our domain is natural numbers but we want to say something about all primes, we simply write $\forall n : (\text{prime}\,(n) \Rightarrow \ldots)$. Don't be fooled; this formula is in no way suggesting that all numbers are prime!

Note: This same construct using $\exists$ is usually a mistake. Consider $\exists x : (P(x) \Rightarrow \ldots)$. By choosing x to be any non-P element, this entire formula is true, without even glancing at what is inside the "…"!

Note: If you have to demonstrate that all ravens are black, $\forall i : (\text{isRaven}\,(i) \Rightarrow \text{isBlack}\,(i))$, there are two ways to do so: You can go out and find every raven and verify that it's black. Alternately, you can go and find every non-black item, and verify that it's a non-raven. Epistemologists, philosophers dealing with how we humans come to learn and know things (about, say, raven colors), go on to ponder about real-world degrees-of-belief: If we have only looked at some ravens, and we find another raven and confirm it is black, does this increase our degree of

belief about all ravens being black? If so, then whenever we find a non-black item which is a non-raven, this must also increase our degree of belief that all ravens are black. This leads to Hempel's (so-called) Paradox: if we are looking for evidence to choose between two competing hypotheses (say, "all non-black items are not ravens" versus "all non-orange items are not ravens"), then finding a purple cow increases our belief in **both** of these hypotheses, simultaneously!

First-Order Logic: bound variables, free variables

In the previous examples we often re-used variable names, even within the same formula. This shouldn't be surprising or confusing, since we do the same thing in programs (another formal language). In fact, the same notions of **bound** and **free** variables occur in both situations. An occurrence of variable x is bound if it is in the body of a quantifier $\forall x$ … or $\exists x$ …. Otherwise, the occurrence is free.

For example, in $\forall x : (\text{likes}\,(x, y))$, the variable y is free but x is not. So this is a statement about y; we can't evaluate this to true/false until we get some context for y. It's useful as a subpart for some bigger formula.

Note: The concept "x free in φ" does **not** talk about the context of φ. So don't confuse it with "well, over on this part of the page, φ happens to occur as the sub-part of another formula containing $\forall x : (\ldots)$, so x really is bound." (Just as 7 is prime, even though people sometimes use 7 in the context of 7+1.) Whether x is free in a φ can be determined by a function $\text{isFree}(x, \varphi)$, needing no other information to produce an answer.

Looking back at our previous examples, we can see that many of the formulas we made had no free variables — all variables were bound by some quantifier in the formula. The truth of such formulas depends only on the interpretation and not on any additional knowledge about what any free variables refer to. Thus, these formulas are common and important enough that we give them a special name, **sentences**.

A given variable name can actually have **both** bound and free occurrences within the same formula, as in $R(x) \ \wedge \ \exists x : (\neg R(x))$. (This formula about x **is** satisfiable: it says that R is true about x, but isn't true about everything.) In essence, there are two different underlying variables going on, but they each happen to have the same name; from scope it can be decided which one each occurence refers to. In programming language terms, we'd say that the inner x (the local variable) **shadows** the outer x

(the enclosing variable). In these terms, free variables in logic correspond to global variables in programs.

Clearly $\forall x : (R(x))$ is always equivalent to $\forall y : (R(y))$; variable names are entirely arbitrary (except maybe for their mnemonic value). So the previous formula might be more clearly re-written as $R(x) \ \wedge \ \exists y : (\neg R(y))$. (This careful re-writing while respecting a variable's scope is called α-**renaming**.) Even if 17 quantifiers each used the same variable (name) x, we could carefully α-renaming 17 times, and end up with an equivalent formula where all quantifiers use distinct variables. This will be useful to avoid potential confusion, especially in the upcoming inference rules, where we'll be introducing and eliminating quantifiers.

Example:
The formula $\forall x : (A(x)) \ \wedge \ \exists x : (B(x)) \ \wedge \ \forall x : (C(x))$ is equivalent to the more readable $\forall x : (A(x)) \ \wedge \ \exists y : (B(y)) \ \wedge \ \forall z : (C(z))$.

First-Order Logic: normal forms, revisited

CNF and DNF revisited (Optional)

In first-order logic, normal forms are still useful for providing a notion of a canonical form. However, their other benefit of corresponding closely to truth tables does not apply here, since truth tables aren't useful for first-order logic.

A formula in **Prenex Conjunctive Normal Form**, or **Prenex CNF**, has a body in CNF preceded by a series of quantifiers. Similarly, a formula in **Prenex Disjunctive Normal Form**, or **Prenex DNF**, has a body in DNF preceded by a series of quantifiers.

Example:
Assuming φ is in CNF, then the following are each in prenex CNF. On the other hand, if φ is in DNF, these are in prenex DNF.

- φ
- $\forall x.\varphi$
- $\exists x.\forall y.\exists z.\varphi$

Every formula has an equivalent prenex CNF formula and equivalent prenex CNF formula. For brevity, we'll skip proving this.

First-Order Logic: equivalences

Now that we can express interesting concepts using the quantifiers "$\exists$" ("there exists") and "$\forall$" ("for all"), how can we use them for the problem of determining whether a formula is true? Back in lowly propositional logic, we had three methods:

- truth tables,
- equivalences, and
- formal proofs with inference rules.

How can we adapt these approaches, for first-order logic?

Well, truth tables have no analog approach. With quantifiers, we don't have a finite set of propositions. Furthermore, variables can't refer to specific items in the domain until we try to interpret them. And when we do, the domain may be of any size — possibly even infinite. Using a truth table on an infinite domain is clearly infeasible, but the real problem stems from how we want to be able to discuss reasoning without respect to a particular domain.

However, we can add equivalences and inference rules to cope with quantifiers. After showing how to work with quantifiers, we'll come back to examine our newly-augmented systems for those desirable traits, soundness and completeness.

First-order Equivalences

When we upgrade from propositional logic to first-order logic, what changes do we need to make to the laws of boolean algebra? Well first off, we can keep all the existing propositional equivalences. For example, $\forall x : (\neg (\varphi \ \wedge \ \psi)) \equiv \forall x : (\neg\varphi \ \vee \ \neg\psi)$. (Technically, we're even making those equivalences stronger, since those meta-variables φ, ψ, θ can now stand for any **first-order** formula, rather than merely propositional formulas.)

But, we also need **additional** identities to deal with our new-fangled quantifiers. What should these be? The most interesting are those that relate the two kinds of quantifiers. Universal quantification ($\forall$) says that something holds for all members of the domain, and existential quantification ($\exists$) says that something holds for at least one member. Clearly, $\forall x : (\varphi)$ implies $\exists x : (\varphi)$, but the other direction doesn't hold, so that is not an equivalence.

Note: Wait just a minute! That implication holds only if the domain is non-empty, so that there is at least one member in it. We'll see this restriction appear a few times.

What about $\forall x : (\neg\varphi)$? In English, "for all items x, $\varphi(x)$ does not hold". A more natural way to say this is that there is **no** item x such that $\varphi(x)$ **does** hold — that is, $\neg\exists x : (\varphi)$. Indeed, this will be one of our new boolean algebra rules.

See a list of equivalences with quantifiers. As before, we can use these to show other pairs of formulas equivalent, as in the following examples.

Example:
Using these identities, we can simplify formulas such as the following:
$\forall y : (\forall x : (R(x) \ \wedge \ Q(x, y))) \ \wedge \ \neg \exists z : (\neg R(z))$.

1	$\forall y : (\forall x : (R(x) \ \wedge \ Q(x, y))) \ \wedge \ \neg \exists z : (\neg R(z))$	
2	$\equiv \forall y : (\forall x : (R(x) \ \wedge \ Q(x, y))) \ \wedge \ \forall z : (\neg \neg R(z))$	Complementation of $\exists$
3	$\equiv \forall y : (\forall x : (R(x) \ \wedge \ Q(x, y))) \ \wedge \ \forall z : (R(z))$	Double Complementation
4	$\equiv \forall x : (\forall y : (R(x) \ \wedge \ Q(x, y))) \ \wedge \ \forall z : (R(z))$	Reordering $\forall$s
5	$\equiv \forall x : (\forall y : (R(x)) \ \wedge \ \forall y : (Q(x, y))) \ \wedge \ \forall z : (R(z))$	Distribution of $\forall$ over $\wedge$
6	$\equiv \forall x : (R(x) \ \wedge \ \forall y : (Q(x, y))) \ \wedge \ \forall z : (R(z))$	Simplification of $\forall$ (y not free in $R(x)$)
7	$\equiv \forall x : (R(x) \ \wedge \ \forall y : (Q(x, y))) \ \wedge \ \forall x : (R(x))$	renaming
8	$\equiv \forall x : (R(x) \ \wedge \ \forall y : (Q(x, y)) \ \wedge \ R(x))$	Distribution of $\forall$ over $\wedge$
9	$\equiv \forall x : (\forall y : (Q(x, y)) \ \wedge \ R(x) \ \wedge \ R(x))$	Commutativity of $\wedge$
10	$\equiv \forall x : (\forall y : (Q(x, y)) \ \wedge \ R(x) \ \wedge \ R(x))$	Associativity of $\wedge$
11	$\equiv \forall x : (\forall y : (Q(x, y)) \ \wedge \ R(x))$	Idempotency of $\wedge$

Admittedly, some of these steps are rather small and obvious (e.g., our use of commutativity and associativity); we include them to illustrate how the identities of propositional logic are also used in first-order logic.

Example:
An example of $\forall x : (\psi) \equiv \psi$ where ψ doesn't contain x occurring free: Let ψ be the formula we've seen before, asserting that a positive integer n was noncomposite:
$\forall j : (\forall k : ((jk = n) \Rightarrow (j = 1) \ \vee \ (k = 1)))$. Since n occurs free, the truth of this formula depends on the value of n. The formula $\forall x : (\psi)$ really is equivalent to ψ: It's true for exactly the same values of n. The use of x is essentially a bit of a rus, since x plays no part of the meat of the ψ.
However, the following formula is certainly **not** equivalent: $\forall n : (\psi)$. This formula asserts that all elements of the domain are non-composite (and it doesn't depend on choosing a particular interpretation for n). Because n occurred free, we can't use the "simplification of quantifiers" identity on it.
Finally, one more variant: $\forall j : (\psi)$. This **is** equivalent to the original, just like $\forall x : (\psi)$ was. Why? The j that occurs inside ψ is a local variable, and is different from any enclosing bindings' j. As we saw, local variables shadow less-local ones, just as in most programming languages.

Exercise:

Problem:

The equivalences for distributing implication over equivalences seem counterintuitive at first glance. Show that the following one holds, given all the identities which don't involve both implication and quantifiers.

Assuming that ψ does not have any free occurrences of variable x,
$\forall x : (\varphi \Rightarrow \psi) \equiv \exists x : (\varphi) \Rightarrow \psi$.

Solution:

1	$\forall x : (\varphi \Rightarrow \psi)$	
2	$\equiv \forall x : (\neg\varphi \ \vee \ \psi)$	Definition of $\Rightarrow$
3	$\equiv \forall x : (\neg\varphi) \ \vee \ \psi$	Distribution of $\forall$ over $\vee$

4	$\equiv \neg\exists x : (\varphi) \ \vee \ \psi$	Complementation of $\exists$
5	$\equiv \exists x : (\varphi) \Rightarrow \psi$	Definition of $\Rightarrow$

Are the following two sentences true?

- "All flying pigs wear top hats." $\forall p : (\mathrm{wears\_top\_hat}(p))$ (over the domain of flying pigs).
- "All numbers in the empty set are even." $\forall x : (\mathrm{even}\ (x))$ (over the empty domain).
- "Every Pulitzer prize winner I've met thinks I'm smart, and cute, too!" $\forall x : (\mathrm{thinksImSmartAndCute}\ (x))$ (over the empty, since I haven't met any Pulitzer prize winners).

Each sentence states that some property holds for every member of some set (flying pigs or the empty set), but there are no such members. Such sentences are considered **vacuously true**.

Okay, maybe you believe that the sentences aren't false, but you still want some reason to consider them true. Well, think of their negations:

- "There exists a flying pig not wearing a top hat." $\exists p : (\neg \mathrm{wears\_top\_hat})$, over the (empty) domain of flying pigs. You can't go off and find a flying pig which contradicts this, since you can't find any flying pig at all. (Note that the negation **isn't** "No flying pigs wear top hats.")
- "There exists a number in the empty set that is even." $\exists x : (\neg\ \mathrm{even})$, over the empty domain. (The negation **isn't** "No numbers in the empty set are even.")

Since these negations are false, the original sentences must be true. This is also similar to the fact that a simple propositional implication, $a \Rightarrow b$ is true, if a is in fact false, regardless of the truth of b; in this crude analogy, a corresponds to "in the domain".

Note: In boolean algebra, we only allow the values false and true, with no third option. This is sometimes called **the law of the excluded middle**. Philosophers **have** developed "trimodal" logics which have a third option, but everything in those logics can be translated into something in traditional logic; such logics might be more convenient in some cases, but they aren't more expressive. **¶ Fuzzy Logic**, on the other hand, is a variant where every proposition has a **degree** of truth (from zero to one). While this **is** different than propositional logic (and, it is the right way to model many real-world problems), as a logic it hasn't yielded interesting new mathematical results.

Even more silliness can ensue when the domain is empty: For example, not only is every member of the empty set even, but every member is simultaneously odd! That is, $\forall x : (R(x) \ \wedge \ \neg R(x))$ is true (only) when the domain is the empty set. Even more degnerately, $\forall x : (\mathrm{false})$ is a true (only) on the emtpy domain.

Are we done yet?

While equivalences are very useful, we are often interested in implications such as the one mentioned previously: $\forall x : (\varphi) \Rightarrow \exists x : (\varphi)$. We could rephrase that as an equivalence, $\forall x : (\varphi) \Rightarrow \exists x : (\varphi) \equiv \mathrm{true}$. Informally, it should be clear that that is rather awkward, and formally it is as well.

But such implications are exactly what inference rules are good for. So, let's continue and consider what first-order inference rules should be.

First-Order Logic: inference rules

Inference with quantifiers

Proving first-order sentences with inference rules is not too different than for propositional ones. We have two slight twists to add: upgrading propositions to relations, and quantifiers. We still keep all our original propositional inference rules, but declare they can now be used on **first-order** WFFs. For our quantifiers, we introduce new first-order inference rules for adding and eliminating quantifiers from formulas. These four new rules look surprisingly simple, but they do have a couple of subtleties we have to keep track of.

Exists-intro

What is the most natural way to prove an existential sentence, like "there exists a prime number greater than 5"? That's easy — you just mention such a number, like 11, and show that it is indeed prime and greater than 5. In other words, once we prove $(11 > 5) \ \wedge \ \forall j : (\forall k : ((11 = jk) \Rightarrow (j = 1) \ \vee \ (k = 1)))$ we can then conclude — using the inference rule **∃Intro** — that the formula $\exists p : ((p > 5) \ \wedge \ \forall j : (\forall k : ((p = jk) \Rightarrow (j = 1) \ \vee \ (k = 1))))$ is true. In general, to prove a formula of the form $\exists x : (\varphi)$, we show that φ holds when x is replaced with some particular **witness**. (The witness was 11 in this example.) The inference rule is $\varphi[p \mapsto c] \vdash \exists p : (\varphi)$. The notation "$\varphi[v \mapsto w]$" means the formula φ but with every occurrence of v replaced by w. For example, we earlier wrote down the formula $\varphi[p \mapsto 11]$, and then decided that this was sufficient to conclude $\exists p : (\varphi)$.

Note: Observe that **you'll never use the substitution-notation "$\varphi[\ldots \mapsto \ldots]$" as part of a literal formula** — it is only used in the inference rule, as a shorthand to describe the actual formula. (It's a pattern-matching metalanguage!)

Note: While it seems like substitution should be a simple textual search-and-replace, it is sometimes more complicated. In the formula $\varphi = (x > 5) \ \wedge \ \exists x : (R(x))$, we don't want $\varphi[x \mapsto 6]$ to try to mention $R(6)$, much less generate something nonsensical like $\forall 6 : (\ldots)$. In programming languages, we say we want "hygienic macros", to respect our the language's notions of variables and scope. E.g., the C pre-processor's `#define` and `#include` notably does **not** respect hygiene, and can inadvertently lead to hard-to-find bugs. Solution: For simplicity, we will always consistently rename variables so that each quantifier binds a distinct variable.

How do you find a witness? That's the difficult part. You, the person creating the proof, must grab a suitable example out of thin air, based on your knowledge of what you want to prove about it. In our previous example, we used our knowledge about prime numbers and about the greater-than relation to pick a witness that would work. In essence, we figured out what facts needed to be true about the witness for the formula to hold, and used that to guide our choice of witness. Of course, this can easily be more difficult, as when proving that there exists a prime greater than 6971 of the form $4x - 1$. (It turns out that 796751 will suffice as a witness here.) Another approach is trial-and-error: Pick some candidate value, and see if it does indeed witness what you're trying to prove. If you succeed, you're done. If not, pick another candidate.

Exists-Elim

The complementary ∃Elim rule corresponds to giving a (new) name to a witness. Thus if you know "there exists some prime bigger than 5", then by ∃Elim we can think of giving some witness the name (say) c, and end up concluding "c is a prime bigger than 5". The caveats are that c must be a new name not already used in the proof, and different from any variables free in the conclusion we're aiming for. However, we will be able to use that variable c along with universal formulas to get useful statements.

Thus the general form of the rule is that $\exists p : (\varphi) \vdash \varphi[p \mapsto c]$. That is, we can rewrite the body of the exists, replacing the quantified variable p with any new variable name c, subject to the restrictions just mentioned.

Forall-Intro

Can we extend that idea to proving a universal sentence? One witness is certainly not enough. We'd need to work with **lots** of witnesses, in fact, every single member of our domain. That's not very practical, especially with infinitely large domains. We need to show that no matter what domain element you choose, the formula holds.

Consider the statements “If n is prime, then we know that …” and “A person X who runs a business should always …”. **Which** n is being talked about, and **which** person? Well, any number or person, respectively. After learning about quantifiers, you may want to preface these sentences with “For all n” or “For all [any] persons X”. But a linguist might point out that while yes “for all” is **related** to the speaker's thought, they are actually using a subtly different mode — that of referring to a single person or number, albeit an anonymous, **arbitrary** one. If “an arbitrary element” really is a natural mode of thought, should our proof system reflect that?

If we choose an **arbitrary** member of the domain, and show that the sentence holds for it, that is sufficient. But, what do we mean by “arbitrary”? In short, it means that we have no control over what element is picked, or equivalently, that the proof must hold regardless of what element is picked. More precisely, a variable is **arbitrary unless**:

- A variable is not arbitrary if it is free in (an enclosing) premise.
- A variable is not arbitrary if it is free after applying ∃Elim — either as the introduced witness c, or free anywhere else in the formula.

The usual way to introduce arbitrary variables is during ∀Elim (w/o later using it in ∃Elim). The formal inference rule for introduction of universal quantification will use these cases as restrictions.

Forall-Elim

Getting rid of universal quantifiers is easy: if you know $\forall x : (\varphi)$ (where φ is a formula presumably involving x), well then you can replace x with anything you want, and the resulting formula will be true. We say $\forall x : (\varphi) \vdash \varphi[x \mapsto t]$ where t is any term. Any variables in t are arbitrary, unless it is an already-existing non-arbitrary variable.

For example, suppose we know that $\forall n : (\text{prime}\,(n) \ \wedge \ (n > 2) \Rightarrow \text{odd}\,(n))$. We can replace n with some term like $m + 4$ to conclude $\text{prime}\,(m + 4) \ \wedge \ (m + 4 > 2) \Rightarrow \text{odd}\,(m + 4)$. The variable m is arbitrary, unless it already occurred in non-arbitrary in a previous line of the proof (perhaps introduced via ∃Elim). A more usual step is to use a term which is just a single variable, and (by coincidence) happens to have the same name as the quantified variable we are eliminating. Thus we often conclude $\text{prime}\,(n) \ \wedge \ (n > 2) \Rightarrow \text{odd}\,(n)$ (note the absence of the initial ∀); n is arbitrary (unless it had already been confusingly in use as a non-arbitrary variable earlier). This is helpful when we'll be later re-introducing the ∀ in a later step; see the example below.

Formal inference rules and proofs

Recall the syllogisms from a previous lecture. The general form of a syllogism is

1. $\forall x : (P(x) \Rightarrow Q(x))$ [major premise]
2. $P(c)$ [minor premise]
3. $Q(c)$ [conclusion]

In our system, we don't have syllogism as a separate rule of inference, but it's easy to see how to translate any syllogism into our system: (for specific relations P and Q, and a specific constant c).

1	$\forall x : (P(x) \Rightarrow Q(x))$	Premise
2	$P(c)$	Premise
3	$P(c) \Rightarrow Q(c)$	∀Elim, by line 1, with $x = c$
4	$Q(c)$	⇒Elim, by lines 2,3, with $\varphi = P = c$ and $\psi = Q = c$

Eliminating a quantifier via ∀Elim and ∃Elim is often merely an intermediate step, where the quantifier will be reintroduced later. This moves the quantification from being explicit to implicit, so that we can use other inference rules on the body of the formula. When this is done, it is very important to pay attention to the restrictions on ∀Intro, so that we don't accidentally “prove” anything too strong.

Example:
$\exists x : (\forall y : (\varphi)) \vdash \forall y : (\exists x : (\varphi))$, for the particular case of $\varphi = R(x, y)$ (other cases all similar).

1	$\exists x : (\forall y : (R(x, y)))$	Premise
2	$\forall y : (R(p, y))$	∃Elim, line 1
3	$R(p, q)$	∀Elim, line 2
4	$\exists x : (R(x, q))$	∃Intro, line 3
5	$\forall y : (\exists x : (R(x, y)))$	∀Intro, line 4

Remember that in line 5, for ∀Intro, we must verify that q is arbitrary. It is, since it was introduced in line 3 by ∀Elim, and there hasn't been an intervening ∃Elim between lines 3 and 5.
We **cannot** instead conclude in line 4 that $\forall x : (R(x, q))$ by ∀Intro, since variable p was introduced by ∃Elim in line 2, and therefore not arbitrary.

Exercise:

Problem:

Let's reverse the previous proof goal:
$\forall y : (\exists x : (\varphi)) \vdash \exists x : (\forall y : (\varphi))$, for the particular case of $\varphi = R(x, y)$ (other cases all similar). This statement does **not** hold in general. So, what's the problem with the following "proof"?

1	$\forall y : (\exists x : (R(x, y)))$	Premise
2	$\exists x : (R(x, q))$	∀Elim, line 1
3	$R(p, q)$	∃Elim, line 2
4	$\forall y : (R(p, y))$	∀Intro, line 3
5	$\exists x : (\forall y : (R(x, y)))$	∃Intro, line 4

Solution:

In line 4, ∀Intro requires that variable being generalized, q, be arbitrary. It was introduced in line 2 by ∀Elim, so that's OK. (E.g., we could've used ∀Intro on line 3 to reintroduce the quantifier just

eliminated.) But, q was free when we used ∃Elim on line 3, and this makes the variable no longer arbitrary. Line 3's choice of p may depend on q, and a variable is only arbitrary if it is free of any such constraints.

The ∀Intro principle is actually very familiar. For instance, after having shown $\neg (a \wedge b) \vdash \neg a \vee \neg b$, we then claimed this was really true for **arbitrary** propositions instead of just a,b. (We actually went a bit further, generalizing individual propositions to entire (arbitrary) WFFs φ,ψ. This could only be done because in any particular interpretation, a formula φ will either be true or false, so replacing it by a proposition still preserves the important part of the proof-of-equivalence.)

The ∀Intro is also used in many informal proofs. Consider: "If a number n is prime, then …". This translates to "$\text{prime}\,(n) \Rightarrow \ldots$", where n is arbitrary. We are entirely used to thinking of this as "$\forall n : (\text{prime}\,(n) \Rightarrow \ldots)$" even though "$n$" was introduced as if it were a particular number.

Proofs and programming

We previously saw that the inference rules of propositional logic are closely related to the process of type checking. The same holds here. For example, in many programming languages, we can write a sorting function that works on **any** type of data. It takes two arguments, a comparison function for the type and a collection (array, list, …) of data of that type. The type of the sorting function can then be described as "for all types T, given a function of type (T and T) → T, and data of type (collection T), it returns data of type (collection T)". This **polymorphic** type-rule uses universal quantification.

Note that the details about substitutions and capture noted here arise in any kind of program that manipulates expressions with bound variables. That includes not only automated theorem provers, but compilers. To avoid such issues, many systems essentially rename all variables by using pointers or some similar system of each variable referring to its binding-site.

When people speak of proofs written by computer, they're talking about this style of inference rule proofs.

Exercises for First-Order Logic

Throughout these exercises, $a \neq b$ is simply a shorthand for $\neg (a = b)$.

Relations and Interpretations

Exercise:

Problem: Consider the binary relation is-a-factor-of on the domain $\{1, 2, 3, 4, 5, 6\}$.

1. List all the ordered pairs in the relation.
2. Display the relation as a directed graph.
3. Display the relation in tabular form.
4. Is the relation reflexive? symmetric? transitive?

Exercise:

Problem: How would you define addsTo as a ternary relation?

1. Give a prose definition of $\text{addsTo}\,(x, y, z)$ in terms of the addition function.
2. List the set of triples in the relation on the domain $\{1, 2, 3, 4\}$.

Exercise:

Problem:

Generalize the previous problem to describe how you can represent any k-ary function as a $(k + 1)$-ary relation.

Exercise:

Problem:

Are each of the following formulas valid, i.e., true for all interpretations? (Remember that the relation names are just names in the formula; don't assume the name has to have any bearing on their interpretation.)

- For arbitrary a and b in the domain, $\text{atLeastAsWiseAs}\,(a, b) \ \vee \ \text{atLeastAsWiseAs}\,(b, a)$
- For arbitrary a in the domain, $\text{prime}\,(a) \Rightarrow (\text{odd}\,(a) \Rightarrow \text{prime}\,(a))$
- For arbitrary a and b in the domain, $\text{betterThan}\,(a, b) \Rightarrow \neg\,\text{betterThan}\,(b, a)$

For each, if it is true or false under all interpretations, prove that. For these small examples, a truth table like this one will probably be easier than using Boolean algebra or inference rules. Otherwise, give an interpretation in which it is true, and one in which it is false.

Note: As always, look at trivial and small test cases first. Here, try domains with zero, one, or two elements, and small relations.

Exercise:

Problem:[Practice problem—solution provided.]

Suppose we wanted to represent the count of neighboring pirates with a binary relation, such that when location A has two neighboring pirates, $\text{piratesNextTo}\,(A, 2)$ will be true. Of course, $\text{piratesNextTo}\,(A, 1)$ would not be true in this situation. These would be analogous with the propositional WaterWorld propositions A-has-2 and A-has-1, respectively.

1. If we only allow binary relations to be subsets of a domain crossed with itself, then what must the domain be for this new relation piratesNextTo?

2. If we further introduced another relation, isNumber?, what is a formula that would help distinguish intended interpretations from unintended interpretations? That is, give a formula that is true under all our intended interpretations of piratesNextTo but is not true for some "nonsense" interpretations we want to exclude. (This will be a formula without an analog in the WaterWorld domain axioms.)

Solution:

1. The relation needs to accept locations as well as numbers, so the domain is $L \cup \mathbb{N}$, where L is the set of WaterWorld locations. Alternatively, you could use $\{0, 1, 2, 3\}$ instead of $\mathbb{N}$, the set of all natural numbers.

2. The difficulty is that it's possible to ask about nonsensical combinations like $\text{piratesNextTo}\,(17, 2)$ and $\text{piratesNextTo}\,(W, B)$. Adding isNumber?, any interpretation would be expected to satisfy, for arbitrary a and b, $\text{piratesNextTo}\,(a, b) \Rightarrow \text{isNumber?}(b) \ \wedge \ \neg\text{isNumber?}(a, b)$.

> **Note:** More interestingly though, imagine we did interpret piratesNextTo over the domain $\mathbb{N}$ only. We could then pretend that the locations, instead of being named A,…,Z, were just numbered 1,…,24. While this representation doesn't reflect how we model the problem, it is legal. Exercise for the reader: Write a formula which excludes relation piratesNextTo which can't match this convention!

Exercise:

Problem:

Determine whether the relation R on the set of all people is reflexive, antireflexive, symmetric, antisymmetric, and/or transitive, where $(a, b) \in R$ if and only if …

1. a is older than b.
2. a is at least as old as b.
3. a and b are exactly the same age.
4. a and b have a common grandparent.
5. a and b have a common grandchild.

Exercise:

Problem:

For each of the following, if the statement is true, explain why, and if the statement is false, give a counter-example relation.

1. If R is reflexive, then R is symmetric.
2. If R is reflexive, then R is antisymmetric.
3. If R is reflexive, then R is not symmetric.
4. If R is reflexive, then R is not antisymmetric.
5. If R is symmetric, then R is reflexive.
6. If R is symmetric, then R is antireflexive.
7. If R is symmetric, then R is not antireflexive.

Quantifiers

Exercise:

Problem: Let $P(x)$ be the statement "has been to Prague", where the domain consists of your classmates.

1. Express each of these quantifications in English.
   - $\exists x : (P(x))$
   - $\forall x : (P(x))$
   - $\neg\exists x : (P(x))$
   - $\neg\forall x : (P(x))$
   - $\exists x : (\neg P(x))$
   - $\forall x : (\neg P(x))$
   - $\neg\exists x : (\neg P(x))$
   - $\neg\forall x : (\neg P(x))$
2. Which of these mean the same thing?

Exercise:

Problem:

Let $C(x)$ be the statement "x has a cat", let $D(x)$ be the statement "x has a dog", and let $F(x)$ be the statement "x has a ferret". Express each of these statements in first-order logic using these relations. Let the domain be your classmates.

1. A classmate has a cat, a dog, and a ferret.
2. All your classmates have a cat, a dog, or a ferret.
3. At least one of your classmates has a cat and a ferret, but not a dog.

4. None of your classmates has a cat, a dog, and a ferret.

5. For each of the three animals, there is a classmate of yours that has one.

Exercise:

Problem:

Determine the truth value of each of these statements if the domain is all real numbers. Where appropriate, give a witness.

1. $\exists x : \left(x^2 = 2\right)$
2. $\exists x : \left(x^2 = -1\right)$
3. $\forall x : \left(x^2 + 2 \geq 1\right)$
4. $\forall x : \left(x^2 \neq x\right)$

Interpreting First-order Formulas

Exercise:

Problem:

Let $P(x)$, $Q(x)$, $R(x)$, and $S(x)$ be the statements "x is a duck", "x is one of my poultry", "x is an officer", and "x is willing to waltz", respectively. Express each of these statements using quantifiers, logical connectives, and the relations $P(x)$, $Q(x)$, $R(x)$, and $S(x)$.

1. No ducks are willing to waltz.

2. No officers ever decline to waltz.

3. All my poultry are ducks.

4. My poultry are not officers.

5. Does the fourth item follow from the first three taken together? Argue informally; you don't need to use the algebra or inference rules for first-order logic here.

Exercise:

Problem:

You come home one evening to find your roommate exuberant because they have managed to prove that there is an even prime number bigger than two. More precisely, they have a correct proof of $\exists y : (P(y) \ \wedge \ (y > 2) \Rightarrow E(y))$, for the domain of natural numbers, with P interpreted as "is prime?" and E interpreted as "is even?". While they are celebrating their imminent fame at this amazing mathematical discovery, you ponder…

1. …and realize the formula is indeed true for that interpretation. Briefly explain why. You don't need to give a formal proof using Boolean algebra or inference rules; just give a particular value for y and explain why it satisfies the body of " $\exists y : (y)$ ".

2. Is the formula still true when restricted to the domain of natural numbers two or less? Briefly explain why or why not.

3. Is the formula still true when restricted to the empty domain? Briefly explain why or why not.

4. Give a formula that correctly captures the notion " there is an even prime number bigger than 2 ".

Exercise:

Problem:

For the sentence $\forall x : (\forall y : (A(x) \ \wedge \ B(x, y) \Rightarrow A(y)))$ state whether it is true or false, relative to the following interpretations. If false, give values for x and y witnessing that.

1. The domain of the natural numbers, where A is interpreted as "even?", and B is interpreted as "equals"
2. The domain of the natural numbers, where A is interpreted as "even?", and B is interpreted as "is an integer divisor of"
3. The domain of the natural numbers, where A is interpreted as "even?", and B is interpreted as "is an integer multiple of"
4. The domain of the Booleans, $\{\mathrm{true}, \mathrm{false}\}$, where A is interpreted as "false?", and B is interpreted as "equals"
5. The domain of WaterWorld locations in the particular board where locations Y and Z contain pirates, but all other locations are safe, the relation symbol A is interpreted as "unsafe?", and B is interpreted as "neighbors"
6. All WaterWorld boards, where A is interpreted as "safe?", and B is interpreted as "neighbors". (That is, is the formula valid for WaterWorld?)

Exercise:

Problem:

Translate the following conversational English statements into first-order logic, using the suggested predicates, or inventing appropriately-named ones if none provided. (You may also freely use $=$ which we'll choose to always interpret as the standard equality relation.)

1. "All books rare and used". This is claimed by a local bookstore; what is the intended domain? Do you believe they mean to claim "all books rare **or** used"?
2. " Everybody who knows that UFOs have kidnapped people knows that Agent Mulder has been kidnapped. " (Is this true, presuming that no UFOs have actually visited Earth…yet?)

Exercise:

Problem:

Write a formula for each of the following. Use the two binary relations isFor and $\mathrm{isAgainst}$ and domain of all people.

- "All for one, and one for all!" We'll take "one" to mean "one particular person", and moreover, that both "one"s are referring the same particular person, resulting in "There is one whom everybody is for, and that one person is for everybody." [footnote]
  Dumas' original musketeers presumably meant something different: that **each** one of them was for each (other) one of the them, making the vice-versa clause redundant. But this is boring for our situation, so we'll leave that interpretation to Athos, Porthos, and Aramis alone.)
- "If you're not for us, you're against us." In aphorisms, "you" is meant to be an arbitrary person; consider using the word "one" instead. Furthermore, we'll interpret "us" as applying to everybody. That is, " One always believes that `if one is not for me, then one is against me' ".
- "The enemy of your enemy is your friend." By "your enemy" we mean "somebody you are against", and similarly, "your friend" will mean "somebody you are for". (Be carefule! This may be different than "somebody who is against/for you").
- "Somebody has an enemy." (We don't know of an aphorism expressing this. [footnote])

None of the following **quite** capture it: "Life's not a bed of roses"; "It's a dog-eat-dog world"; "Everyone for themselves"; "You can't please all the people all the time".

Two interpretations are considered fundamentally the same (or **isomorphic**) if you can map one interpretation to the other simply by a consistent renaming of domain elements.

Exercise:

Problem:[Practice problem—solution provided.]

Find two fundamentally different interpretations that satisfy the statement "There exists one person who is liked by two people".

Solution:

One interpretation that satisfies this is a domain of three people Alice, Bob, Charlie, with the likes relation: $\{(\text{Alice}, \text{Bob}), (\text{Bob}, \text{Bob})\}$. Bob is liked by two people, so it satisfies the statement.

Here's another interpretation that is the same except for renaming, and thus **not** fundamentally different: a domain of three people Alyssa, Bobby, Chuck, with the likes relation: $\{(\text{Chuck}, \text{Alyssa}), (\text{Alyssa}, \text{Alyssa})\}$. With the substitutions $[\text{Chuck} \mapsto \text{Alice}]$ and $[\text{Alyssa} \mapsto \text{Bob}]$, we see that the underlying structure is the same as before.

Here's an interpretation that **is** fundamentally different: a domain of three people Alice, Bob, Charlie, with the likes relation: $\{(\text{Charlie}, \text{Bob}), (\text{Alice}, \text{Bob})\}$. No matter how you rename, you don't get somebody liking themself, so you can see its underlying structure is truly different than the preceding interpretations.

English is fuzzy enough that it is unclear whether "one" and "two" are meant as exact counts. The above two examples each assumed they are.

Note: If we change the statement slightly to add a comma: " There exists one person, who is liked by two people ", we arguably change the meaning significantly. The now-independent first clause arguably means there is only one person existent in total, so the overall statement must be false! There's a quick lesson in the difference between English dependent and independent clauses.

Exercise:

Problem:

For the four "Musketeer" formulas from a previous exercise, find three fundamentally different interpretations of isFor which satisfy all the formulas on a domain of three people.

Depict each of these interpretations as a **graph**. Draw three circles (**nodes**) representing the three people, and an arrow (**edge**) from a person to each person they like. (You can glance at Rosen Section 9.1, Figure 8 for an example.)

Note: One of the interpretations is unintuitive in that isFor and isAgainst don't correspond to what we probably mean in English.

Exercise:

Problem:

Translate the following statements into first-order logic. The domain is the set of natural numbers, and the binary relation $\mathrm{kth}\,(k, n)$ indicates whether or not the kth number of the sequence is n. For example, the sequence $(5, 7, 5)$, is represented by the relation $\mathrm{kth} = \{(0, 5), (1, 7), (2, 5)\}$. You can also use the binary relations $=$, $<$, and $\leq$, but no others.

You may assume that kth models a sequence. No index k is occurs multiple times, thus excluding $\mathrm{kth} = \{(0, 5), (1, 7), (0, 9)\}$. Thus, kth is a function, as in a previous example representing an array as a function. Also, no higher index k occurs without all lower-numbered indices being present, thus excluding $\mathrm{kth} = \{(0, 5), (1, 7), (3, 9)\}$.

1. The sequence is finite.
2. The sequence contains at least three distinct numbers , e.g., $(5, 6, 5, 6, 7, 8)$, but not $(5, 6, 5, 6)$.

3. The sequence is sorted in non-decreasing order, e.g., $(3, 5, 5, 6, 8, 10, 10, 12)$.
4. The sequence is sorted in non-decreasing order, except for exactly one out-of-order element, e.g., $(20, 30, 4, 50, 60)$.

Exercise:

Problem:

Some binary relations can be viewed as the encoding of a unary function, where the first element of the ordered pair represents the function's value. For instance, in a previous exercise we encoded the binary function addition as a ternary relation addsTo.

1. Give one example of a binary relation which does **not** correspond to the encoding of a function.

   Write a first-order formula describing the properties that a binary relation R must have to correspond to
2. a unary function.

Exercise:

Problem:

Alternation of quantifiers: Determine the truth of each of the following sentences in each of the indicated domains.

Note: To help yourself, you might want to develop an English version of what the logic sentences say. Start with the inner formula (talking about people x,y,z), then add the quantifier for z to get a statement about people x,y, and repeat for the other two quantifiers.

Four sentences:

1. $\forall x : (\forall y : (\exists z : (\mathrm{likes}\,(x, y) \ \wedge \ ((z \neq y) \Rightarrow \neg\,\mathrm{likes}\,(y, z)))))$
2. $\exists x : (\forall y : (\forall z : (\mathrm{likes}\,(x, y) \ \wedge \ ((z \neq y) \Rightarrow \neg\,\mathrm{likes}\,(y, z)))))$
3. $\exists x : (\exists y : (\forall z : (\mathrm{likes}\,(x, y) \ \wedge \ ((z \neq y) \Rightarrow \neg\,\mathrm{likes}\,(y, z)))))$
4. $\forall x : (\exists y : (\forall z : (\mathrm{likes}\,(x, y) \ \wedge \ ((z \neq y) \Rightarrow \neg\,\mathrm{likes}\,(y, z)))))$

Four domains:

1. The empty domain.
2. A world with one person, who likes herself.
3. A world with Yorick and Zelda, where Yorick likes Zelda, Zelda likes herself, and that's all.
4. A world with many people, including CJ (Catherine Zeta-Jones), JC (John Cusack), and JR (Julia Roberts). Everybody likes themselves; everybody likes JC; everybody likes CJ except JR; everybody likes JR except CJ and IB. Any others may or may not like each other, as you choose, subject to the preceding. (You may wish to sketch a graph of this likes relation, similar to Rosen Section 9.1 Figure 8.)

Determine the truth of all sixteen combinations of the four statements and four domains.

Modeling

Exercise:

Problem:

Translate the following into first-order logic: " Raspberry sherbet with hot fudge (rshf) is the tastiest dessert. " Use $\mathrm{tastier}$ as your only relation.

What is the intended domain for your formula? What is a relation which makes this statement true? One which makes it false?

Exercise:

Problem:

Even allowing for ellision, the list of WaterWorld domain axioms is incomplete, in a sense. The game reports how many pirates exist in total, but that global information is not reflected in the propositions or axioms. We had the same problem with the propositional logic domain axioms

1. First, assume we only use the default WaterWorld board size and number of pirates, i.e., five. What additional axiom or axioms do we need?
2. Next, generalize your answer to model the program's ability to play the game with a different number of pirates. What problem do you encounter?

Exercise:

Problem:

The puzzle game of Sudoku is played on a 9×9 grid, where each square holds a number between 1 and 9. The positions of the numbers must obey constraints. Each row and each column has each of the 9 numbers. Each of the 9 non-overlapping 3×3 square sub-grids has each of the 9 numbers.

Like WaterWorld, throughout the game, some of the values have not been discovered, although they are determined. You start with some numbers revealed, enough to guarantee that the rest of the board is uniquely determined by the constraints. Thus, like in WaterWorld, when deducing the value of another location, what has been revealed so far would serve as premises in a proof.

Fortunately, there are the same number of rows, columns, subgrids, and values. So, our domain is $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

To model the game, we will use the following relations:

- $\mathrm{value}\,(r, c, v)$ indicates that at row r, column c is the value v.
- $v = w$ is the standard equality relation.
- $\mathrm{subgrid}\,(g, r, c)$ indicates that subgrid g includes the location at row r, column c.

Provide domain axioms for Sudoku, and briefly explain them. These will model the row, column, and subgrid constraints. In addition, you should include constraints on our above relations, such as that each location holds one value.

Reasoning with Equivalences

Exercise:

Problem:

Some of the first-order equivalences are redundant. For each of the following, prove the equivalence using the other equivalences.

1. $\forall x : (\phi \Rightarrow \theta) \equiv \exists x : (\phi) \Rightarrow \theta$
2. Assuming a non-empty domain, $\exists x : (\theta \Rightarrow \phi) \equiv \theta \Rightarrow \exists x : (\phi)$.

Exercise:

Problem:

We can characterize a prime number as a number n satisfying $\forall q : (\forall r : ((qr = n) \Rightarrow (q = 1) \ \vee \ (r = 1)))$. Using the equivalences for first-order logic, show step-by-step that this is equivalent to the formula $\neg\exists q : (\exists r : ((qr = n) \ \wedge \ (q \neq 1) \ \wedge \ (r \neq 1)))$. Do **not** use any arithmetic equivalences.

Exercise:

Problem:

A student claims that $\forall x : (A(x) \ \wedge \ B(x) \Rightarrow C(z)) \equiv \forall x : (A(x)) \ \wedge \ \forall x : (B(x)) \Rightarrow C(z)$ by the "distribution of quantifiers". This is actually trying to do two steps at once. Rewrite this as the two separate intended steps, determine which is wrong, and describe why that step is wrong.

Exercise:

Problem:

Simplify the formula $\forall x : (\forall y : (\exists z : (A(x) \ \wedge \ B(y) \Rightarrow C(z))))$, so that the body of each quantifier contains only a single atomic formula involving that quantified variable. Provide reasoning for each step of your simplification.

Reasoning with Inference Rules

Exercise:

Problem:[Practice problem—solution provided.]

Prove that syllogisms are valid inferences. In other words, show that $\forall x : (R(x) \Rightarrow S(x)), R(c) \vdash S(c)$.

Solution:

1	$\forall x : (R(x) \Rightarrow S(x))$	Premise

2	$R(c)$		Premise
3	$R(c) \Rightarrow S(c)$		∀Elim, line 1
4	$X(c)$		⇒Elim, lines 2,3

Exercise:

Problem: What is wrong with the following "proof" of $\exists x : (E(x)) \Rightarrow E(c)$?

1	subproof:$\exists x : (E(x)) \vdash E(c)$			
1.a		$\exists x : (E(x))$	Premise for subproof	
1.b		$E(c)$	∃Elim, line 1.a	
2	$\exists x : (E(x)) \Rightarrow E(c)$			⇒Intro, line 1

Exercise:

Problem:

Using the inference rules, **formally** prove the last part of the previous problem about ducks and such.

Exercise:

Problem:

Give an inference rule proof of
$\forall x : (\text{Fruit}\,(x) \Rightarrow \text{hasMethod}\,(\text{tasty}, x)), \forall y : (\text{Apple}\,(y) \Rightarrow \text{Fruit}\,(y)) \vdash \forall z : (\text{Apple}\,(z) \Rightarrow \text{hasMethod}\,(\text{t}$
.

Exercise:

Problem:

1. Prove the following: $\exists x : (P(x)), \forall y : (P(y) \Rightarrow Q(y)) \vdash \exists z : (Q(z))$
2. Your proof above used ∃Intro. Why can't we replace that step with the formula $\forall z : (Q(z))$ with the justification "∀Intro"?
3. Describe an interpretation which satisfies the proof's premises, but does not satisfy $\forall z : (Q(z))$.

Logic: Looking Back
A recap of where we've been, and why we traveled there.

Why didn't we begin with quantifiers all along?

We saw three stages of logics:

- Propositional logic, with formulas like $\mathrm{DickLikesJane} \Rightarrow \neg\,\mathrm{JaneLikesDick}$. While the propositions are named suggestively, nothing in the logic enforces a relation among these; it is equivalent to $A \Rightarrow \neg B$.
- Predicate logic, where variables (and constants) can express a connection between different parts of the formula: $\mathrm{likes}\,(y, x) \Rightarrow \neg\,\mathrm{likes}\,(x, y)$ Predicate logic introduced the idea of variables, and required domains and interpretations to determine truth. But it can't bind variables, and thus requires an interpretation of x and y to evaluate.
- First-order logic, which included two quantifiers to bind variables: $\forall y : (\exists x : (\mathrm{likes}\,(y, x) \Rightarrow \neg\,\mathrm{likes}\,(x, y)))$

So why, you might ask, didn't we just start out with first-order logic in the first lecture? One reason, clearly, is to introduce concepts one at a time: everything you needed to know about one level was needed in the next, and then some. But there's more: by restricting our formalisms, we can't express all the concepts of the bigger formalism, but we **can** have automated ways of checking statements or finding proofs.

In general, this is a common theme in the theory of any subject: determining when and where you can (or, need to) trade off expressibility for predictive value. For example, …

- Linguistics: Having a set of precise rules for (say) Tagalog grammar allows you to determine what is and isn't a valid sentence; details of the formal grammar can reveal relations to other languages which aren't otherwise so apparent. On the other hand, a grammar for any natural language is unlikely to exactly capture all things which native

speakers say and understand. If working with a formal grammar, one needs to know what is being lost and what is being gained.

    - Dismissing a grammar as irrelevant because it doesn't entirely reflect usage is missing the point of the grammar;
    - Conversely, condemning some real-life utterances as ungrammatical (and ignoring them) forgets that the grammar is a model which captures many (if not all) important properties.

  Of course, any reasonable debate on this topic respects these two poles and is actually about where the best trade-off between them lies.
- Psychology: Say, Piaget might propose four stages of learning in children. It may not trade off total accuracy, for (say) clues of what to look for in brain development.
- Physics: Modern pedagogy must trade off quantum accuracy for Newtonian approximations. Researchers exploring fields like particle physics must trade off exact simulations for statistical ("stochastic") approximations.

Understanding the theoretical foundations of a field is often critical for knowing how to apply various techniques in practice.

Logic and everyday reasoning

We've looked at the impreciseness and ambiguity of natural language statements, but these are not the only problems hidden in natural language arguments. The following illustrates a common form of hidden assumption: saying "the tenth reindeer of Santa Claus is …" implies the existence some tenth reindeer. More subtly, humans use much more information than what is spoken in a conversation. Even aside from body language, consider a friend asking you "Hey, are you hungry?" While as a formal statement this doesn't have any information, in real life it highly suggests that your friend **is** hungry.

A much more blatant form of missing information is when the speaker simply chooses to omit it. When arguing for a cause it is standard practice

to simply describe its advantages, without any of its disadvantages or alternatives.

> **Note:** Economists measure things not in cost, but **opportunity cost**, the price of something minus the benefits of what you'd get using the price for something else. E.g., for $117 million the university can build a new research center. But what else could you do on campus with $117m?

Historically, logic and **rhetoric**, the art of persuasion through language, are closely linked.

Other logics

You've now been introduced to two logics: propositional and first-order. But, the story does not have to end here. There are many other logics, each with their uses.

Limitations of first-order logic's expressiveness

We can make first-order sentences to express concepts as " vertices a and b are connected by a path of length 2 ", as well as "…by a path of length 3", " length ≤ 4 ", etc.

> **Note:** Write a couple of these sentences!

But trying to write " vertices a and b are connected [by a path of any length] " isn't obvious … in fact, it can be proven that **no** first-order sentence can

express this property! Nor can it express the closely-related property "the graph is connected" (without reference to two named vertices a and b).

Hmm, what about **second**-order logic? It has a bigger name; whatever it means, perhaps it can express more properties?

What exactly is **second-order logic**? In first-order logic, quantifiers range over elements of the domain: " there exist numbers x and y, … ". In second-order logic, you can additionally quantify over **sets** of elements of the domain: " there is a set of numbers, such that … ".

Example:
For instance, " for all vertices x and y, there exists a set of vertices (call the set 'Red'), the red vertices include a path from x to y ". More precisely, " every Red vertex has exactly two Red neighbors, or it is x or y (which each have exactly 1 red neighbor) ". Is this sentence true exactly when the graph is connected? Why does this description of "red vertices" not **quite** correspond to " just the vertices on a path from x to y "?

An interesting phenomenon: There are some relations between how difficult it is to write down a property, and how difficult to compute it! How might you try to formalize the statement "there is a winning strategy for chess"?

A shortcoming of first-order logic is that it is **impossible** to express the concept "path". (This can be proven, though we won't do so here.)

Thus, some other logics used to formalize certain systems include:

- As mentioned, second-order logic is like first-order logic, but it also allows quantification over entire **relations**. Thus, you can make formulas that state things like " For all relations R, if R is symmetric and transitive, then … ". While less common, we could continue with third-order, fourth-order, etc.

- **Temporal logic** is based on quantification over time. This is useful to describe how a program's state changes over time. In particular, it is used for describing concurrent program specifications and communication protocols, sequences of communications steps used in security or networking. See, for example, TeachLogic's Model-Checking module.
- **Linear logic** is a "resource-aware" logic. Every premise must be used, but it may be used only once. This models, for example, how keyboard input is usually handled: reading an input also removes it from the input stream, so that it can't be read again.

Logic in computer science

Logics provide us with a formal language useful for

- specifying properties unambiguously,
- proving that programs and systems do (or don't) have the claimed properties, and
- gaining greater insight into other languages such as database queries.

Programming language type systems are a great example of these first two points. The connectives allow us to talk about pairs and structures (x **and** y), unions (x **or** y), and functions (**if** you give the program a x, it produces a y). The "generics" in Java, C++, and C# are based upon universal quantification, while "wildcards" in Java are based upon existential quantification. One formalization of this strong link between logic and types is called the **Curry-Howard isomorphism**.

Compilers have very specific logics built into them. In order to optimize your code, analyses check what properties your code has e.g., are variables b and c needed at the same time, or can they be stored in the same hardware register?

More generally, it would be great to be able to verify that our hardware and software designs were correct. First, specifying what "correct" means requires providing the appropriate logical formulas. With hardware, automated verification is now part of the regular practice. However, it is so

computationally expensive that it can only be done on pieces of a design, but not, say, a whole microprocessor. With software, we also frequently work with smaller pieces of code, proving individual functions or algorithms correct. However, there are two big inter-related problems. Many of the properties we'd like to prove about our software are "undecidable" — it is **impossible** to check the property accurately for every input. Also, specifying full correctness typically requires extensions to first-order logic, most of which are incomplete. [footnote] As we've seen, that means that we **cannot** prove everything we want. While proving hardware and software correct has its limitations, logic provides us with tools that are still quite useful. For an introduction to one approach used in verification, see TeachLogic's Model-Checking module.

Even something as simple as first-order logic using the integers as our domain and addition and multiplication as relations is undecidable. *Kurt Gödel, 1931*

Acknowledgements

The TeachLogic Project is the work of many contributors, and was made possible through an NSF CISE grant. Major contributors and grant Principle Investigators are

- Moshe Vardi, Rice University
- Matthias Felleisen, Northeastern University
- Ian Barland, Rice University
- Phokion Kolaitis, University of California at Santa Cruz
- John Greiner, Rice University

In addition, Paul Steckler implemented the Base module's Waterworld game.

Students who helped contribute to various TeachLogic modules include (chronologically)

- Peggy Fidelman
- Justin Garcia
- Brian Cohen
- Sarah Trowbridge
- Bryan Cash
- Fuching “Jack” Chi
- Ben McMahan

TeachLogic has also been influenced by the Beseme project, headed by Rex Page of Oklahoma University; in particular the Base module owes both some overall structure and specific details to Beseme.

Janice Bordeaux, from the Engineering Dean's office at Rice University, assisted with developing classroom assessment tools.

Reference: propositional equivalences

The following lists some propositional formula equivalences. Remember that we use the symbol $\equiv$ as a relation between two WFFs, not as a connective inside a WFF. In these, φ, ψ, and θ are meta-variables standing for any WFF.

Double Complementation	$\neg\neg\varphi \equiv \varphi$	
Complement	$\varphi \lor \neg\varphi \equiv \text{true}$	$\varphi \land \neg\varphi \equiv \text{false}$
Identity	$\varphi \lor \text{false} \equiv \varphi$	$\varphi \land \text{true} \equiv \varphi$
Dominance	$\varphi \lor \text{true} \equiv \text{true}$	$\varphi \land \text{false} \equiv \text{false}$
Idempotency	$\varphi \lor \varphi \equiv \varphi$	$\varphi \land \varphi \equiv \varphi$
Absorption	$\varphi \land (\varphi \lor \psi) \equiv \varphi$	$\varphi \lor \varphi \land \psi \equiv \varphi$
Redundancy	$\varphi \land (\neg\varphi \lor \psi) \equiv \varphi \land \psi$	$\varphi \lor \neg\varphi \land \psi \equiv \varphi \lor \psi$
DeMorgan's Laws	$\neg(\varphi \land \psi) \equiv \neg\varphi \lor \neg\psi$	$\neg(\varphi \lor \psi) \equiv \neg\varphi \land \neg\psi$
Associativity	$\varphi \land (\psi \land \theta) \equiv (\varphi \land \psi) \land \theta$	$\varphi \lor (\psi \lor \theta) \equiv (\varphi \lor \psi) \lor \theta$
Commutativity	$\varphi \land \psi \equiv \psi \land \varphi$	$\varphi \lor \psi \equiv \psi \lor \varphi$
Distributivity	$\varphi \land (\psi \lor \theta) \equiv \varphi \land \psi \lor \varphi \land \theta$	$\varphi \lor \psi \land \theta \equiv (\varphi \lor \psi) \land (\varphi \lor \theta)$

Propositional Logic Equivalences

Equivalences for implication are omitted above for brevity and for tradition. They can be derived, using the definition $a \Rightarrow b \equiv \neg a \lor b$.

Example:
For example, using Identity and Commutativity, we have
$\text{true} \Rightarrow b \equiv \neg\text{true} \lor b \equiv \text{false} \lor b \equiv b \lor \text{false} \equiv b$.

Reference: propositional inference rules

Abbreviation	**Name**	**If you know all of…**	**…then you can infer**
$\wedge$Intro	and-introduction	φ ψ	$\varphi \wedge \psi$
$\wedge$Elim	and-elimination (left)	$\varphi \wedge \psi$	φ
	and-elimination (right)	$\varphi \wedge \psi$	ψ
$\vee$Intro	or-introduction (left)	φ	$\varphi \vee \psi$
	or-introduction (right)	ψ	$\varphi \vee \psi$

Abbreviation	**Name**	**If you know all of…**	**…then you can infer**
$\vee$Elim	or-elimination	$\varphi \vdash \theta$ $\psi \vdash \theta$ $\varphi \ \vee \ \psi$	θ
$\Rightarrow$Intro	if-introduction	$\varphi,\psi,\ldots,\theta \vdash \omega$	$\varphi \ \wedge \ \psi \ \wedge \ \ldots \ \wedge \ \theta \Rightarrow \omega$
$\Rightarrow$Elim	if-elimination (modus ponens)	$\varphi \Rightarrow \psi$ φ	ψ
falseIntro	false-introduction	φ $\neg\varphi$	false

Abbreviation	**Name**	**If you know all of…**	**…then you can infer**
falseElim	false-elimination	false	φ
RAA	reductio ad absurdum (v. 1)	$\neg\varphi \vdash \text{false}$	φ
	reductio ad absurdum (v. 2)	$\varphi \vdash \text{false}$	$\neg\varphi$
¬Intro	negation-introduction	φ	$\neg\neg\varphi$
¬Elim	negation-elimination	$\neg\neg\varphi$	φ
CaseElim	case-elimination (left)	$\varphi \vee \psi$ $\neg\varphi$	ψ
	case-elimination (right)	$\varphi \vee \psi$ $\neg\psi$	φ

Our propositional inference rules

As usual, φ, ψ, θ, ω are meta-variables standing for any WFF.

This is by no means the only possible inference system for propositional logic.

> **Note:** This set of inference rules is based upon *Discrete Mathematics with a Computer* by Hall and O'Donnell (Springer, 2000) and The Beseme Project.

first-order equivalences
Some equivalences for manipulation of first-order formulas.

The following equivalences are in addition to those of propositional logic. In these, φ and ψ each stand for any WFF, but θ **stands for any WFF with no free occurrences of** x .

Equivalence	**∀ Variant**	**∃ Variant**
Complementation of Quantifiers	$\forall x : (\neg\varphi) \equiv \neg\exists x : (\varphi)$	$\exists x : (\neg\varphi) \equiv \neg\forall x : (\varphi)$
Interchanging Quantifiers	$\forall x : (\forall y : (\varphi)) \equiv \forall y : (\forall x : (\varphi))$	$\exists x : (\exists y : (\varphi)) \equiv \exists y : (\exists x : (\varphi))$
Distribution of Quantifiers	$\forall x : (\varphi \ \wedge \ \psi) \equiv \forall x : (\varphi) \ \wedge \ \forall x : (\psi)$	$\exists x : (\varphi \ \vee \ \psi) \equiv \exists x : (\varphi) \ \vee \ \exists x : (\psi)$
	$\forall x : (\varphi \ \vee \ \theta) \equiv \forall x : (\varphi) \ \vee \ \theta$	$\exists x : (\varphi \ \wedge \ \theta) \equiv \exists x : (\varphi) \ \wedge \ \theta$
	$\forall x : (\varphi \Rightarrow \theta) \equiv \exists x : (\varphi) \Rightarrow \theta$	
	$\forall x : (\theta \Rightarrow \varphi) \equiv \theta \Rightarrow \forall x : (\varphi)$	
Distribution of Quantifiers — with non-empty domain	$\forall x : (\varphi \ \wedge \ \theta) \equiv \forall x : (\varphi) \ \wedge \ \theta$	$\exists x : (\varphi \ \vee \ \theta) \equiv \exists x : (\varphi) \ \vee \ \theta$
		$\exists x : (\varphi \Rightarrow \theta) \equiv \forall x : (\varphi) \Rightarrow \theta$
		$\exists x : (\theta \Rightarrow \varphi) \equiv \theta \Rightarrow \exists x : (\varphi)$
Renaming	$\forall x : (\varphi) \equiv \forall y : (\varphi[x \mapsto y])$	$\exists x : (\varphi) \equiv \exists y : (\varphi[x \mapsto y])$
Simplification of Quantifiers — with non-empty domain	$\forall x : (\theta) \equiv \theta$	$\exists x : (\theta) \equiv \theta$
Simplification of Quantifiers — with empty domain	$\forall x : (\varphi) \equiv \mathrm{true}$	$\exists x : (\varphi) \equiv \mathrm{false}$

First-order Logic Equivalences

When citing Distribution of Quantifiers, say what you're distributing over what: e.g., " distribute ∀ over ∨ (with θ being x-free) ".

In renaming, the notation $\varphi[x \mapsto y]$ means " φ with each free occurrence of x replaced by y ". It is a meta-formula; when writing any particular formula you don't write any brackets, and instead just do the replacement.

This set of equivalences isn't actually quite complete. For instance,
$\exists x : (\forall y : (R(x, y))) \Rightarrow \forall y : (\exists x : (R(x, y)))$ is equivalent to true, but we can't show it using only the rules

above. It does become complete[footnote] if we add some analogs of the first-order inference rules, replacing $\vdash$ with $\Rightarrow$ (and carrying along their baggage of "arbitrary" and "free-to-substitute-in").
It's not obvious when this system is complete; that's Gödel's completeness theorem, his 1929 Ph.D. thesis. Don't confuse it with his more celebrated **In**completeness Theorem, on the other hand, which talks about the ability to prove formulas which are true in all interpretations which include arithmetic (as opposed to **all** interpretations everywhere.)

Reference: first-order inference rules

The following are in addition to those of propositional logic.

Abbreviation	Name	If you know all of…	…then you can infer
∀Intro	∀-introduction	φ y arbitrary.	$\forall x.\varphi[y \mapsto x]$

Abbreviation	Name	If you know all of…	…then you can infer
∀Elim	∀-elimination	$\forall x.\varphi$ t is any term that is free to be replaced in φ. Domain non-empty.	$\varphi[x \mapsto t]$

Abbreviation	Name	If you know all of…	…then you can infer
∃Intro	∃-introduction	φ t is any term in φ that is free to be replaced. Domain non-empty.	$\exists x.\varphi[t \mapsto x]$, where t is arbitrary

Abbreviation	Name	If you know all of…	…then you can infer
∃Elim	∃-elimination	$\exists x.\varphi$ c is a new constant in the proof. c does not occur in the proof's conclusion.	$\varphi[x \mapsto c]$

Our first-order inference rules

As usual, we use φ as a meta-variable to range over first-order WFFs. Similarly, t is a meta-variable for first-order terms, and c is a meta-variable for domain constants. The notation $\varphi[v \mapsto w]$ means the formula φ but with every appropriate occurrence of v replaced by w.

As discussed in the lecture notes, a variable is **arbitrary unless**:

- A variable is not arbitrary if it is free in (an enclosing) premise.
- A variable is not arbitrary if it is free after applying ∃Elim — either as the introduced witness c, or free anywhere else in the formula.

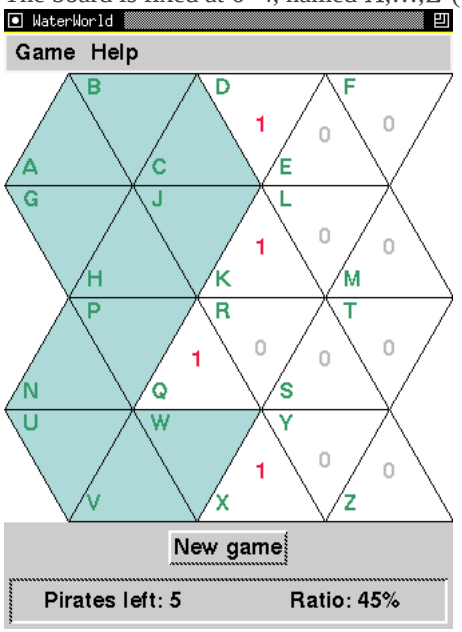
The usual way to introduce arbitrary variables is during ∀Elim (w/o later using it in ∃Elim).

As a detail in $\forall$Elim and $\exists$Intro, the term t must be **free to replace** the variable x in φ. This means that it is **not** the case that both t contains a variable quantified in φ, and that x occurs free within that quantifier. In short, the bound variable names should be kept distinct from the free variable names. Also, only free occurrences x get replaced. The restriction in $\exists$Elim on c being new is similar.

Reference: propositional WaterWorld

We summarize the details of how we choose to model WaterWorld boards in propositional logic: exactly what propositions we make up, and the formal domain axioms which capture the game's rules.

The board is fixed at 6×4, named A,…,Z (with I and O omitted).



A Sample WaterWorld board

Propositions

There are a myriad of propositions for WaterWorld, which can be grouped:

- Whether or not a location contains a pirate: A-unsafe, B-unsafe, …, Z-unsafe.
- Whether or not a location contains no pirate: A-safe, B-safe, …, Z-safe.

> **Note:** Yes, using the intended interpretation, these are redundant with the previous ones. Some domain axioms below will formalize this.

- Propositions indicating the number of neighboring pirates, to a location: A-has-0, A-has-1, A-has-2, B-has-0, B-has-1, B-has-2, …, H-has-0, H-has-1, H-has-2, H-has-3, …, Z-has-0, Z-has-1. These are all true/false propositions; — there are no explicit numbers in the logic. A domain axiom below will assert that whenever (say) B-has-1 is true, then B-has-0 and B-has-2 are both false.

> **Note:** There is no proposition A-has-3 — since location A has only two neighbors. Similarly, there is no proposition B-has-3. We **could** have chosen to include those, but under the intended interpretation they'd always be false.

These propositions describe the state of the underlying board — the model — and not our particular view of it. Our particular view will be reflected in which formulas we'll accept as premises. So we'll accept A-has-2 as a premise only when A has been exposed and shows a 2.

The domain axioms

Axioms asserting that the neighbor counts are correct:

- Count of 0:
  - "A0": $\text{A-has-0} \Rightarrow \text{B-safe} \ \wedge \ \text{G-safe}$
  - …
  - "H0": $\text{H-has-0} \Rightarrow \text{G-safe} \ \wedge \ \text{J-safe} \ \wedge \ \text{P-safe}$
  - …
  - "Z0": $\text{Z-has-0} \Rightarrow \text{Y-safe}$
- Count of 1:
  - "A1": $\text{A-has-1} \Rightarrow \text{B-safe} \ \wedge \ \text{G-unsafe} \ \vee \ \text{B-unsafe} \ \wedge \ \text{G-safe}$
  - …
  - "H1":
    $\text{H-has-1} \Rightarrow \text{G-safe} \ \wedge \ \text{J-safe} \ \wedge \ \text{P-unsafe} \ \vee \ \text{G-safe} \ \wedge \ \text{J-unsafe} \ \wedge \ \text{P-safe} \ \vee \ \text{G-unsafe} \ \wedge \ \text{J-sa}$
  - …
  - "Z1": $\text{Z-has-1} \Rightarrow \text{Y-unsafe}$
- Count of 2:
  - "A2": $\text{A-has-2} \Rightarrow \text{B-unsafe} \ \wedge \ \text{G-unsafe}$
  - …
  - "H2":
    $\text{H-has-2} \Rightarrow \text{G-safe} \ \wedge \ \text{J-unsafe} \ \wedge \ \text{P-unsafe} \ \vee \ \text{G-unsafe} \ \wedge \ \text{J-safe} \ \wedge \ \text{P-unsafe} \ \vee \ \text{G-unsafe} \ \wedge$
  - …

  There aren't any such axioms for locations with only one neighbor.
- Count of 3:
  - "H3": $\text{H-has-3} \Rightarrow \text{G-unsafe} \ \wedge \ \text{J-unsafe} \ \wedge \ \text{P-unsafe}$
  - …

  There aren't any such axioms for locations with only one or two neighbors.

Axioms asserting that the propositions for counting neighbors are consistent:

- $\text{A-has-0} \ \vee \ \text{A-has-1}$
- $\text{A-has-0} \Rightarrow \neg\text{A-has-1}$
- $\text{A-has-1} \Rightarrow \neg\text{A-has-0}$
- $\text{B-has-0} \ \vee \ \text{B-has-1} \ \vee \ \text{B-has-2}$
- $\text{B-has-0} \Rightarrow \neg\text{B-has-1} \ \wedge \ \neg\text{B-has-2}$
- $\text{B-has-1} \Rightarrow \neg\text{B-has-0} \ \wedge \ \neg\text{B-has-2}$
- $\text{B-has-2} \Rightarrow \neg\text{B-has-0} \ \wedge \ \neg\text{B-has-1}$
- …
- $\text{H-has-0} \ \vee \ \text{H-has-1} \ \vee \ \text{H-has-2} \ \vee \ \text{H-has-3}$
- $\text{H-has-0} \Rightarrow \neg\text{H-has-1} \ \wedge \ \neg\text{H-has-2} \ \wedge \ \neg\text{H-has-3}$
- $\text{H-has-1} \Rightarrow \neg\text{H-has-0} \ \wedge \ \neg\text{H-has-2} \ \wedge \ \neg\text{H-has-3}$
- $\text{H-has-2} \Rightarrow \neg\text{H-has-0} \ \wedge \ \neg\text{H-has-1} \ \wedge \ \neg\text{H-has-3}$
- $\text{H-has-3} \Rightarrow \neg\text{H-has-0} \ \wedge \ \neg\text{H-has-1} \ \wedge \ \neg\text{H-has-2}$
- …

Axioms asserting that the safety propositions are consistent:

- $\text{A-safe} \Rightarrow \neg\text{A-unsafe}$,
- $\neg\text{A-safe} \Rightarrow \text{A-unsafe}$,
- $\ldots$
- $\text{Z-safe} \Rightarrow \neg\text{Z-unsafe}$,
- $\neg\text{Z-safe} \Rightarrow \text{Z-unsafe}$.

This set of axioms is not quite complete, as explored in an exercise.

As mentioned, it is redundant to have both A-safe and A-unsafe as propositions. Furthermore, having both allows us to express inconsistent states (ones that would contradict the safety axioms). If implementing this in a program, you might use both as variables, but have a safety-check function to make sure that a given board representation is consistent. Even better, you could implement WaterWorld so that these propositions wouldn't be variables, but instead be calls to a lookup (accessor) functions. These would examine the same internal state, to eliminate the chance of inconsistent data.

Using only true/false propositions; without recourse to numbers makes these domain axioms unwieldy. Later, we'll see how relations and quantifiers help us model the game of WaterWorld more concisely.

Reference: first-order WaterWorld

We summarize the details of how we choose to model WaterWorld boards in first-order logic: exactly what relations we make up, and the formal domain axioms which capture the game's rules.

This will follow almost exactly the same pattern as our WaterWorld model in propositional logic. However, we will take advantage of the additional flexibility provided by first-order logic.

Rather than modeling only the default 6×4 WaterWorld board;, we will be able to model any board representable by our relations. This will allow boards of any size and configuration, with one major constraint — each location can have at most three neighboring pirates.

Domain and Relations

Our domain is simply the set of all board locations. This set can be arbitrarily large — even infinite!

The board configuration is given by the binary "neighbor" relation nhbr.

The next relations correspond directly to the propositions in the propositional logic model.

- Whether or not a location contains a pirate: safe. This is a unary relation.

> **Note:** We choose not to include a redundant relation unsafe.

- Unary relations indicating the number of neighboring pirates: $\mathrm{has0}$, $\mathrm{has1}$, $\mathrm{has2}$, and $\mathrm{has3}$.

> **Note:** Thus, we have our restriction to three unsafe neighbors. This will also be reflected in our domain axioms below. See also this problem for a discussion of how to avoid this restriction.

In addition, to have encode the domain axioms for an arbitrary domain, we also need an equality relation over our domain of locations. As is traditional, we will use infix notation for this relation, for example, $x = y$. Furthermore, we will allow ourselves to write $x \neq y$ as shorthand for $\neg\,(x = y)$. Thus, we do not need a distinct inequality relation.

Note that these relations describe the state of the underlying board — the model — and not our particular view of it. Our particular view will be reflected in which formulas we'll accept as premises. So we'll accept $\mathrm{has2}\,(A)$ as a premise only when A has been exposed and shows a 2.

The domain axioms

Many of our axioms correspond directly, albeit much more succinctly, with those of the propositional model. In addition, we have axioms that specify that our neighbor and equality relations are self-consistent.

Axioms asserting that the neighbor relation is anti-reflexive and symmetric:

- $\forall x : (\neg\,\mathrm{nhbr}\,(x, x))$
- $\forall x : (\forall y : (\mathrm{nhbr}\,(x, y) \Rightarrow \mathrm{nhbr}\,(y, x)))$

Axioms asserting that "=" truly is an equality relation, i.e., it is reflexive, symmetric, and transitive.

- $\forall x : (x = x)$
- $\forall x : (\forall y : ((x = y) \Rightarrow (y = x)))$
- $\forall x : (\forall y : (\forall z : ((x = y) \ \wedge \ (y = z) \Rightarrow (x = z))))$

Axioms asserting that the neighbor counts are correct. Each of these is of the form "if location x has n neighboring pirates, then there are n **distinct** unsafe neighbors of x, and any **other distinct** neighbor x is safe." We use the equality relation to specify the distinctness of each neighbor.

- $\forall x : (\mathrm{has0}\,(x) \Rightarrow \forall y : (\mathrm{nhbr}\,(x, y) \Rightarrow \mathrm{safe}\,(y)))$
- $\forall x : (\mathrm{has1}\,(x) \Rightarrow \exists a : (\mathrm{nhbr}\,(x, a) \ \wedge \ \neg\,\mathrm{safe}\,(a) \ \wedge \ \forall y : (\mathrm{nhbr}\,(x, y) \ \wedge \ (a \neq y) \Rightarrow \mathrm{safe}\,(y))))$
- $\forall x : (\mathrm{has2}\,(x) \Rightarrow \exists a : (\exists b : (\mathrm{nhbr}\,(x, a) \ \wedge \ \mathrm{nhbr}\,(x, b) \ \wedge \ (a \neq b) \ \wedge \ \neg\,\mathrm{safe}\,(a) \ \wedge \ \neg\,\mathrm{safe}\,(b) \ \wedge \ \forall y : (\mathrm{nhbr}\,(x, y) \ \wedge \ (y \neq a) \ \wedge \ (y \neq b) \Rightarrow \mathrm{safe}\,(y)))))$
- $\forall x : (\mathrm{has3}\,(x) \Rightarrow \exists a : (\exists b : (\exists c : (\mathrm{nhbr}\,(x, a) \ \wedge \ \mathrm{nhbr}\,(x, b) \ \wedge \ \mathrm{nhbr}\,(x, c) \ \wedge \ (a \neq b) \ \wedge \ (a \neq c) \ \wedge \ (b \neq c) \ \wedge \ \forall y : (\mathrm{nhbr}\,(x, y) \ \wedge \ (y \neq a) \ \wedge \ (y \neq b) \ \wedge \ (y \neq c) \Rightarrow \mathrm{safe}\,(y))))))$

In addition, we want the implications to go the opposite way. Otherwise, each of has0, has1, has2, and has3 could always be false, while still satisfying the above! For brevity, we elide the details in the following list:

- $\forall x : (\forall y : (\mathrm{nhbr}\,(x, y) \Rightarrow \mathrm{safe}\,(y)) \Rightarrow \mathrm{has0}\,(x))$
- $\forall x : (\ldots \Rightarrow \mathrm{has1}\,(x))$
- $\forall x : (\ldots \Rightarrow \mathrm{has2}\,(x))$
- $\forall x : (\ldots \Rightarrow \mathrm{has3}\,(x))$

Axioms asserting that the neighbor counts are consistent. While redundant, including axioms like the following can be convenient.

- $\forall x : (\mathrm{has0}\,(x) \Rightarrow \neg\,(\mathrm{has1}\,(x) \ \vee \ \mathrm{has2}\,(x) \ \vee \ \mathrm{has3}\,(x)))$
- $\forall x : (\mathrm{has1}\,(x) \Rightarrow \neg\,(\mathrm{has0}\,(x) \ \vee \ \mathrm{has2}\,(x) \ \vee \ \mathrm{has3}\,(x)))$
- $\forall x : (\mathrm{has2}\,(x) \Rightarrow \neg\,(\mathrm{has0}\,(x) \ \vee \ \mathrm{has1}\,(x) \ \vee \ \mathrm{has3}\,(x)))$
- $\forall x : (\mathrm{has3}\,(x) \Rightarrow \neg\,(\mathrm{has0}\,(x) \ \vee \ \mathrm{has1}\,(x) \ \vee \ \mathrm{has2}\,(x)))$

Note that this set of axioms is not quite complete, as explored in an exercise.

Browser support

Note: The information in this module is outdated. Please see my course for a table of contents.

Note: This page meant to be viewed with a MathML-enabled browser. If you see (∀x. (P(**x**) → (∃y. (P(**y**) ∨ φ)))) as a nice version of (forall x . (P(x) -> (exists y . (P(y) v phi)))) you're doing okay; If you further see $\mathscr{A} \vdash \mathcal{B}$ as a nice version of (scriptA |- scriptB) you're set! If not, see our description of browser support .

At Rice on the CSNet, use `mozilla`. Preferably, use version 1.1, as currently available on Solaris 8 machines. On frosty.cs, version 1.1 is the default. On other Solaris 8 machines, version 1.1 is not yet the default, but available via `/opt1/mozilla-1.1/sunos5/bin/mozilla`.

In general, to view TeachLogic web pages, you'll need a browser that supports the following features:

- Cascading Style Sheets (CSS) -- Most recent browsers support CSS sufficiently well.
- MathML -- Some browsers support MathML sufficiently well. However, most (all?) do not fully support Unicode Plane 1 numerical entity references, which includes most mathematical alphanumeric characters.
- Math-oriented fonts

Which browsers support these features? The above links provide more details, but here's a summary of some browsers.

- Mozilla and Netscape (version 7.0) -- Both work, except some characters (Unicode Plane 1) don't appear correctly.

- Internet Explorer is not yet an option, even with the MathPlayer plug-in to view MathML. IE won't display pages with some characters (Unicode Plane 1).

Alternatively, PDF versions of the web pages are also provided via the Base module's index .